Regular Expressions

Recap from Last Time

Regular Languages

- A language L is a ***regular language*** if there is a DFA D such that $\mathscr{L}(D) = L$.
- ***Theorem:*** The following are equivalent:
  - L is a regular language.
  - There is a DFA for L.
  - There is an NFA for L.

Language Concatenation

- If $w \in \Sigma^*$ and $x \in \Sigma^*$, then wx is the ***concatenation*** of w and x.
- If L_1 and L_2 are languages over Σ, the ***concatenation*** of L_1 and L_2 is the language L_1L_2 defined as

$$\boldsymbol{L_1L_2 = \{\, wx \mid w \in L_1 \text{ and } x \in L_2 \,\}}$$

Language Exponentiation

- If L is a language over Σ, the language L^n is the concatenation of n copies of L with itself.
  - Special case: $L^0 = \{\varepsilon\}$.
- The ***Kleene closure*** of a language L, denoted L^*, is defined as

$$L^* = \{ w \mid \exists n \in \mathbb{N}.\ w \in L^n \}$$

- Intuitively, all strings that can be formed by concatenating any number of strings in L with one another.

Closure Properties

- ***Theorem:*** If L_1 and L_2 are regular languages over an alphabet Σ, then so are the following languages:
  - $\overline{L}_1$
  - $L_1 \cup L_2$
  - $L_1 \cap L_2$
  - $L_1 L_2$
  - L_1^*
- These properties are called ***closure properties of the regular languages***.

New Stuff!

Another View of Regular Languages

Rethinking Regular Languages

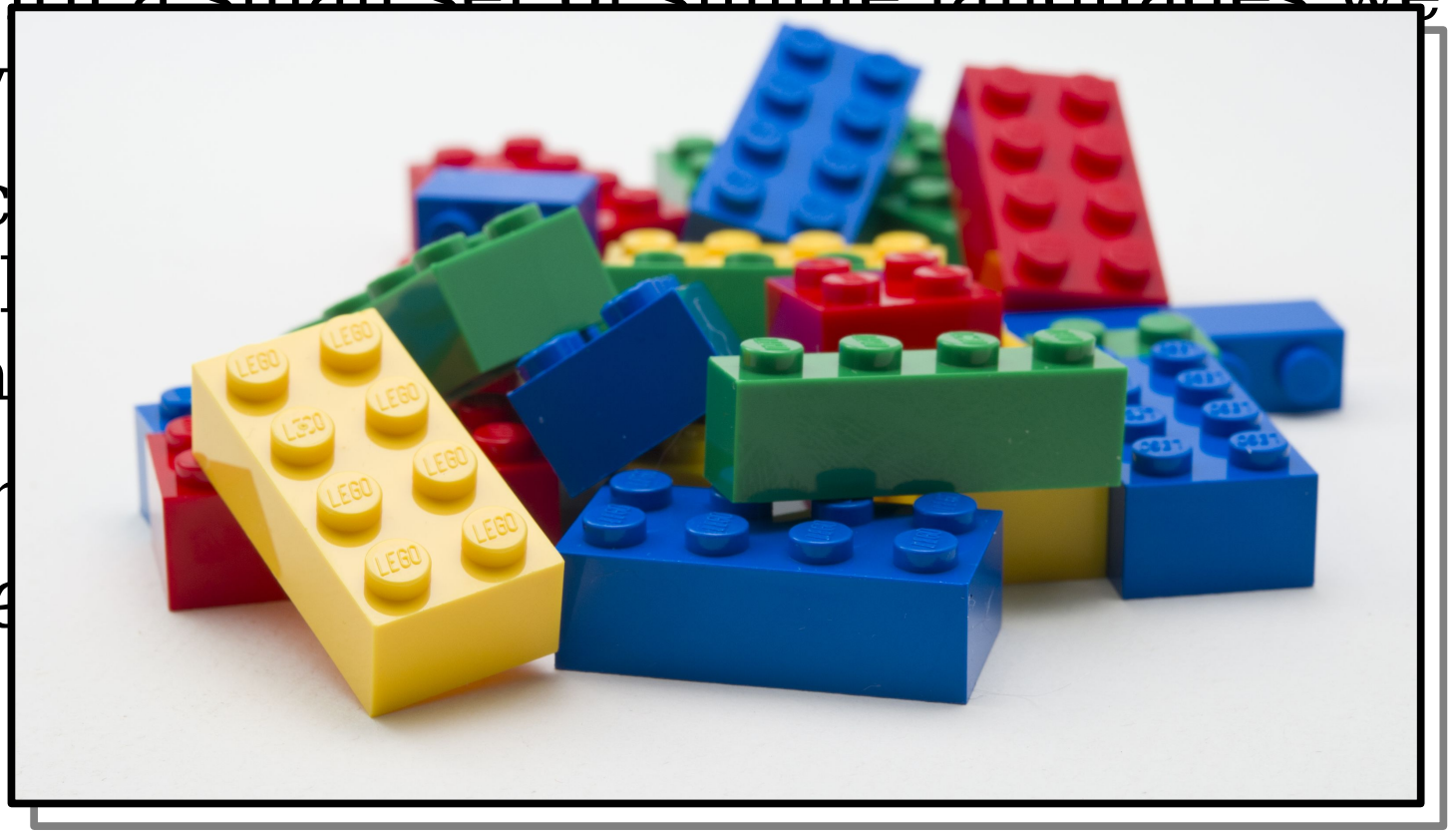
- We currently have several tools for showing a language is regular.
  - Construct a DFA for it.
  - Construct an NFA for it.
  - Apply closure properties to existing languages.
- We have not spoken much of this last idea.

Constructing Regular Languages

- ***Idea:*** Build up all regular languages as follows:
  - Start with a small set of simple languages we already know to be regular.
  - Using closure properties, combine these simple languages together to form more elaborate languages.
- *A bottom-up approach to the regular languages.*

Constructing Regular Languages

- ***Idea:*** Build up all regular languages as follows:
  - Start with a small set of simple languages we already
  - Using c
    simple l
    elabora
- *A bottom*
  language



Regular Expressions

- ***Regular expressions*** are a way of describing a language via a string representation.
- Used extensively in software systems for string processing and as the basis for tools like `grep` and `flex`.
- Conceptually, regular languages are strings describing how to assemble a larger language out of smaller pieces.

Atomic Regular Expressions

- The regular expressions begin with three simple building blocks.
- The symbol **Ø** is a regular expression that represents the empty language Ø.
- For any **a** ∈ Σ, the symbol **a** is a regular expression for the language {**a**}
- The symbol **ε** is a regular expression that represents the language {ε}
  - ***Remember: {ε} ≠ Ø!***
  - ***Remember: {ε} ≠ ε!***

Compound Regular Expressions

- If R_1 and R_2 are regular expressions, $\boldsymbol{R_1R_2}$ is a regular expression for the *concatenation* of the languages of R_1 and R_2.
- If R_1 and R_2 are regular expressions, $\boldsymbol{R_1 \cup R_2}$ is a regular expression for the *union* of the languages of R_1 and R_2.
- If R is a regular expression, $\boldsymbol{R^*}$ is a regular expression for the *Kleene closure* of the language of R.
- If R is a regular expression, $\boldsymbol{(R)}$ is a regular expression with the same meaning as R.

Operator Precedence

- Regular expression operator precedence:

$$(R)$$

$$R*$$

$$R_1 R_2$$

$$R_1 \cup R_2$$

- So **`ab*cUd`** is parsed as **`((a(b*))c)Ud`**

Regular Expression Examples

- The regular expression **`trickUtreat`** represents the regular language { **`trick`**, **`treat`** }
- The regular expression **`booo*`** represents the regular language { **`boo`**, **`booo`**, **`boooo`**, ... }
- The regular expression **`candy!(candy!)*`** represents the regular language { **`candy!`**, **`candy!candy!`**, **`candy!candy!candy!`**, ... }

Regular Expressions, Formally

- The ***language of a regular expression*** is the language described by that regular expression.
- Formally:
  - $\mathscr{L}(\varepsilon) = \{\varepsilon\}$
  - $\mathscr{L}(\emptyset) = \emptyset$
  - $\mathscr{L}(\texttt{a}) = \{\texttt{a}\}$
  - $\mathscr{L}(R_1 R_2) = \mathscr{L}(R_1)\ \mathscr{L}(R_2)$
  - $\mathscr{L}(R_1 \cup R_2) = \mathscr{L}(R_1) \cup \mathscr{L}(R_2)$
  - $\mathscr{L}(R^*) = \mathscr{L}(R)^*$
  - $\mathscr{L}((R)) = \mathscr{L}(R)$

Worthwhile activity: Apply this recursive definition to

a(bUc)((d))

and see what you get.

Regular Expressions are Awesome

- Let $\Sigma = \{$`0`, `1`$\}$
- Let $L = \{ w \in \Sigma^* \mid w$ contains `00` as a substring $\}$

Regular Expressions are Awesome

- Let $\Sigma = \{$`0`, `1`$\}$
- Let $L = \{ w \in \Sigma^* \mid w$ contains `00` as a substring $\}$

$$\mathbf{(0 \cup 1)^*00(0 \cup 1)^*}$$

Regular Expressions are Awesome

- Let $\Sigma = \{0, 1\}$
- Let $L = \{ w \in \Sigma^* \mid w$ contains 00 as a substring $\}$

$$(0 \cup 1)^*00(0 \cup 1)^*$$

Regular Expressions are Awesome

- Let $\Sigma = \{0, 1\}$
- Let $L = \{ w \in \Sigma^* \mid w$ contains 00 as a substring $\}$

$$(0 \cup 1)^*00(0 \cup 1)^*$$

11011100101
0000
11111011110011111

Regular Expressions are Awesome

- Let $\Sigma = \{0, 1\}$
- Let $L = \{ w \in \Sigma^* \mid w$ contains `00` as a substring $\}$

$$(0 \cup 1)^*00(0 \cup 1)^*$$

11011100101
0000
1111101111100011111

Regular Expressions are Awesome

- Let $\Sigma = \{0, 1\}$
- Let $L = \{ w \in \Sigma^* \mid w$ contains 00 as a substring $\}$

Σ*00Σ*

11011100101
0000
1111101111100011111

Regular Expressions are Awesome

- Let $\Sigma = \{$`0`, `1`$\}$
- Let $L = \{\ w \in \Sigma^* \mid |w| = 4\ \}$

Regular Expressions are Awesome

Let $\Sigma = \{0, 1\}$

Let $L = \{ w \in \Sigma^* \mid |w| = 4 \}$

Regular Expressions are Awesome

Let $\Sigma = \{0, 1\}$

Let $L = \{ w \in \Sigma^* \mid |w| = 4 \}$

The length of a string w is denoted $|w|$

Regular Expressions are Awesome

- Let $\Sigma = \{$`0`, `1`$\}$
- Let $L = \{\ w \in \Sigma^* \mid |w| = 4\ \}$

Regular Expressions are Awesome

- Let $\Sigma = \{0, 1\}$
- Let $L = \{ w \in \Sigma^* \mid |w| = 4 \}$

$$\mathbf{\Sigma\Sigma\Sigma\Sigma}$$

Regular Expressions are Awesome

- Let $\Sigma = \{0, 1\}$
- Let $L = \{ w \in \Sigma^* \mid |w| = 4 \}$

$$\Sigma\Sigma\Sigma\Sigma$$

Regular Expressions are Awesome

- Let $\Sigma = \{0, 1\}$
- Let $L = \{ w \in \Sigma^* \mid |w| = 4 \}$

$$\Sigma\Sigma\Sigma\Sigma$$

0000
1010
1111
1000

Regular Expressions are Awesome

- Let $\Sigma = \{0, 1\}$
- Let $L = \{ w \in \Sigma^* \mid |w| = 4 \}$

ΣΣΣΣ

0000
1010
1111
1000

Regular Expressions are Awesome

- Let $\Sigma = \{0, 1\}$
- Let $L = \{ w \in \Sigma^* \mid |w| = 4 \}$

$$\Sigma^4$$

0000
1010
1111
1000

Regular Expressions are Awesome

- Let $\Sigma = \{0, 1\}$
- Let $L = \{ w \in \Sigma^* \mid |w| = 4 \}$

$$\Sigma^4$$

0000
1010
1111
1000

Regular Expressions are Awesome

- Let $\Sigma = \{$`0`, `1`$\}$
- Let $L = \{\ w \in \Sigma^* \mid w$ contains at most one `0` $\}$

Regular Expressions are Awesome

- Let $\Sigma = \{$`0`, `1`$\}$
- Let $L = \{\ w \in \Sigma^* \mid w$ contains at most one `0` $\}$

$1^*(0 \cup \varepsilon)1^*$

Regular Expressions are Awesome

- Let $\Sigma = \{$`0`, `1`$\}$
- Let $L = \{\ w \in \Sigma^* \mid w$ contains at most one `0` $\}$

1*(0 ∪ ε)1*

Regular Expressions are Awesome

- Let $\Sigma = \{0, 1\}$
- Let $L = \{ w \in \Sigma^* \mid w$ contains at most one $0 \}$

$1^*(0 \cup \varepsilon)1^*$

11110111
111111
0111
0

Regular Expressions are Awesome

- Let $\Sigma = \{0, 1\}$
- Let $L = \{ w \in \Sigma^* \mid w$ contains at most one $0 \}$

$1^*(0 \cup \varepsilon)1^*$

11110111
111111
0111
0

Regular Expressions are Awesome

- Let $\Sigma = \{$`0`, `1`$\}$
- Let $L = \{\ w \in \Sigma^* \mid w$ contains at most one `0` $\}$

1*0?1*

11110111
111111
0111
0

Regular Expressions are Awesome

- Let Σ = { **a**, **.**, **@** }, where **a** represents "some letter."
- Regular expression for email addresses:

Regular Expressions are Awesome

- Let Σ = { **a**, **.**, **@** }, where **a** represents "some letter."
- Regular expression for email addresses:

aa*(.aa*)*@aa*.aa*(.aa*)*

Regular Expressions are Awesome

- Let Σ = { `a`, `.`, `@` }, where `a` represents "some letter."
- Regular expression for email addresses:

aa*(.aa*)*@aa*.aa*(.aa*)*

cs103@cs.stanford.edu
first.middle.last@mail.site.org
barack.obama@whitehouse.gov

Regular Expressions are Awesome

- Let Σ = { **a**, **.**, **@** }, where **a** represents "some letter."
- Regular expression for email addresses:

aa*(.aa*)*@aa*.aa*(.aa*)*

cs103@cs.stanford.edu
first.middle.last@mail.site.org
barack.obama@whitehouse.gov

Regular Expressions are Awesome

- Let Σ = { **a**, **.**, **@** }, where **a** represents "some letter."
- Regular expression for email addresses:

aa*(.aa*)*@aa*.aa*(.aa*)*

cs103@cs.stanford.edu
first.middle.last@mail.site.org
barack.obama@whitehouse.gov

Regular Expressions are Awesome

- Let Σ = { **a**, **.**, **@** }, where **a** represents "some letter."
- Regular expression for email addresses:

aa*(.aa*)*@aa*.aa*(.aa*)*

cs103@cs.stanford.edu
first.middle.last@mail.site.org
barack.obama@whitehouse.gov

Regular Expressions are Awesome

- Let Σ = { **a**, **.**, **@** }, where **a** represents "some letter."
- Regular expression for email addresses:

$$a^+ (.aa^*)^*@aa^*.aa^*(.aa^*)^*$$

cs103@cs.stanford.edu
first.middle.last@mail.site.org
barack.obama@whitehouse.gov

Regular Expressions are Awesome

- Let Σ = { **a**, **.**, **@** }, where **a** represents "some letter."
- Regular expression for email addresses:

$$\mathbf{a^+ \ (.a^+)^* \ @ \ a^+.a^+ \ (.a^+)^*}$$

cs103@cs.stanford.edu
first.middle.last@mail.site.org
barack.obama@whitehouse.gov

Regular Expressions are Awesome

- Let Σ = { **a**, **.**, **@** }, where **a** represents "some letter."
- Regular expression for email addresses:

$$\mathbf{a^+ \ (.a^+)^* \ @ \ a^+.a^+ \ (.a^+)^*}$$

cs103@cs.stanford.edu
first.middle.last@mail.site.org
barack.obama@whitehouse.gov

Regular Expressions are Awesome

- Let Σ = { **a**, **.**, **@** }, where **a** represents "some letter."
- Regular expression for email addresses:

$$\mathbf{a^+ \ (.a^+)^* \ @ \quad a^+ \quad (.a^+)^+}$$

cs103@cs.stanford.edu
first.middle.last@mail.site.org
barack.obama@whitehouse.gov

Regular Expressions are Awesome

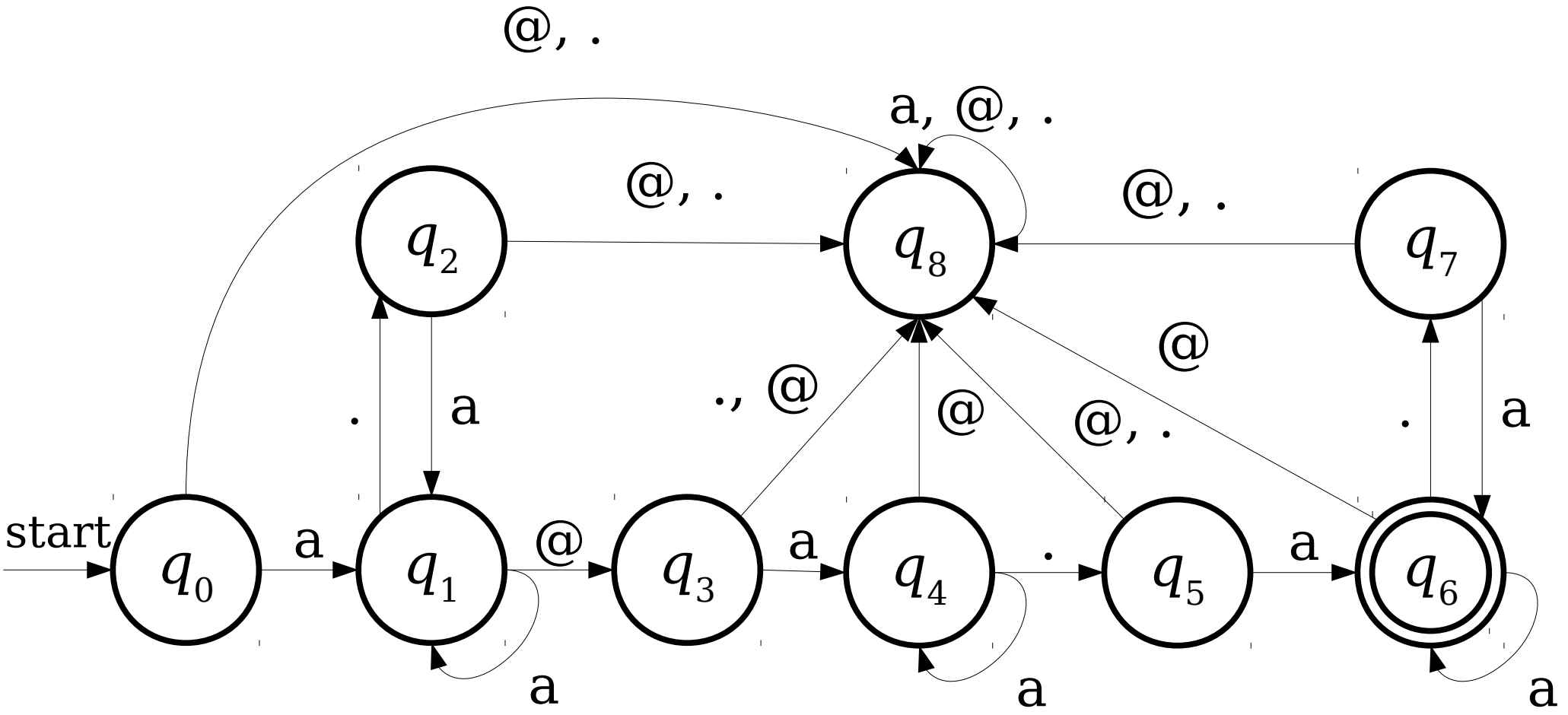
- Let Σ = { **a**, **.**, **@** }, where **a** represents "some letter."
- Regular expression for email addresses:

$$\mathbf{a^+(.a^+)^*@a^+(.a^+)^+}$$

cs103@cs.stanford.edu
first.middle.last@mail.site.org
barack.obama@whitehouse.gov

Regular Expressions are Awesome

$a^+(.a^+)^*@a^+(.a^+)^+$



Shorthand Summary

- R^n is shorthand for $RR \ldots R$ (n times).
  - Edge case: define $R^0 = \varepsilon$.
- Σ is shorthand for "any character in Σ."
- $R?$ is shorthand for $(R \cup \varepsilon)$, meaning "zero or one copies of R."
- R^+ is shorthand for RR^*, meaning "one or more copies of R."

Time-Out for Announcements!

Problem Set Six

- Problem Set Six is due on Friday at the start of class.
  - We have online tools you can use to design your DFAs, NFAs, and regular expressions. We highly recommend using them – they make it super easy to test and submit!
  - By the end of this lecture you'll know everything you need to know for the rest of the problem set.
- Have questions? Stop by office hours or ask on Piazza!

WiCS Casual CS Dinner

- WiCS' Casual CS Dinner is tomorrow from 5PM – 7PM at the WCC.
- Highly recommended – this is one of the best events of the quarter!

CS Career Panel

- The CS Course Advisor has put together an CS Career Panel event for this Thursday from 5:45PM – 7:00PM in the Gates Fifth Floor lounge.
  - There are some *really* cool people on this panel – highly recommended!
- Interested? RSVP at this link: http://goo.gl/forms/A2JEOcQMVN.

Your Questions

“I like CS, and I'm thinking of applying for a coterm, but I'm not sure if I am good enough, and I don't really know any CS professors to the point where they can write me a recommendation. Do you have any advice on either of these?”

Yay! How exciting!

It's extremely common to apply for the coterm without building close connections to professors (think about the average class size). Most of the intro course instructors will be happy to write you a recommendation letter if you did well in their course and have a solid CS GPA.

“Due to how much content we cover in CS103, has it ever been considered to split 103 into two courses? I find that I don't get time to appreciate and process the material fully since we move from one topic to another quite quickly.”

Yep, we've definitely considered this. Believe it or not, this is what the course looks like after we cut about three weeks' worth of material from it.

I doubt this is going to happen anytime soon, though we'll see what happens!

“What kinda music do you like ☺? Anything you'd recommend?”

I tend to like jazz fusion and electronic music (I'm not sure how that happened). A few of my favorite albums:

“The Paris Session” by The Toure-Raichel Collective
“Live at the Quick” by Bela Fleck and the Flecktones
“Random Access Memories” by Daft Punk
“Aja” by Steely Dan
“We Live Here” by Pat Metheny Group
“Tuesday Wonderland” by E.S.T.
“Black Radio” by Robert Glasper

Back to CS103!

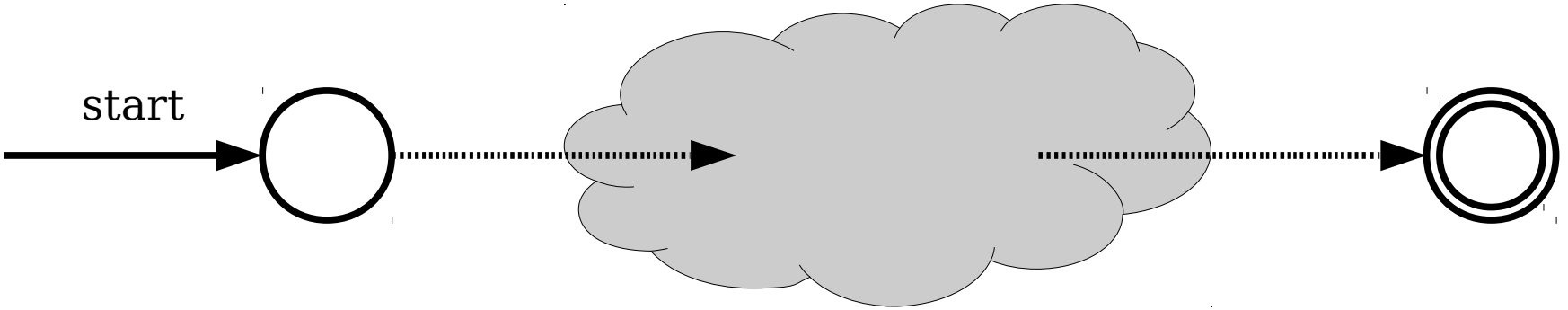
The Power of Regular Expressions

Theorem: If R is a regular expression, then $\mathscr{L}(R)$ is regular.

Proof idea: Show how to convert a regular expression into an NFA.

Thompson's Algorithm

- ***Thompson's algorithm*** is an algorithm for converting any regular expression into an NFA.
- ***Theorem:*** For any regular expression R, there is an NFA N such that
  - $\mathscr{L}(R) = \mathscr{L}(N)$
  - N has exactly one accepting state.
  - N has no transitions into its start state.
  - N has no transitions out of its accepting state.



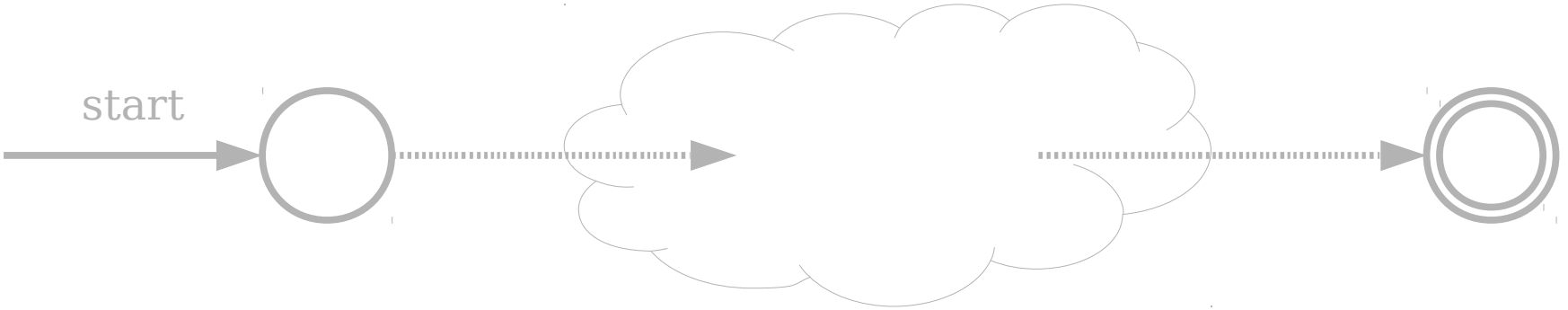
Thompson's Algorithm

Thompson's algorithm is an algorithm for converting any regular expression into an NFA.

Theorem: For any regular expression R, there is an NFA N such that

$$\mathscr{L}(R) = \mathscr{L}(N)$$

- N has exactly one accepting state.
- N has no transitions into its start state.
- N has no transitions out of its accepting state.



Thompson's Algorithm

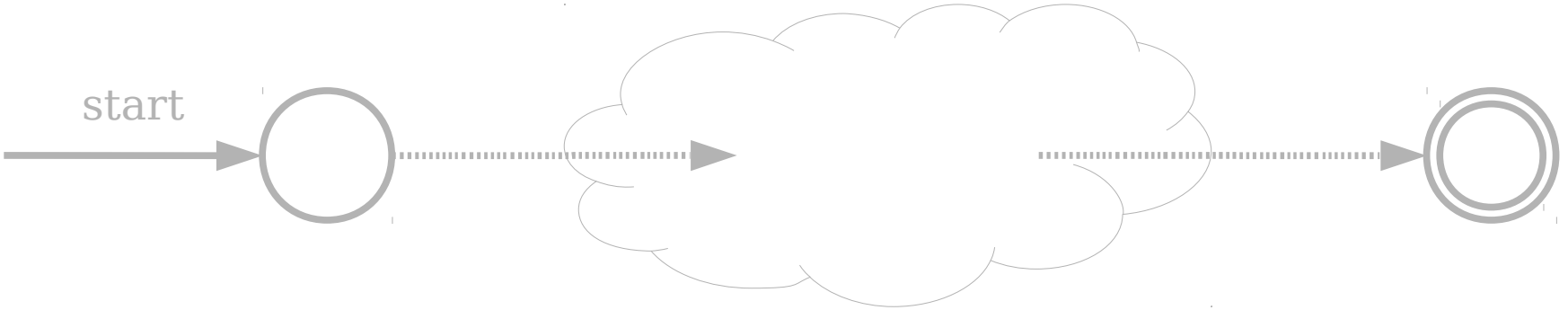
Thompson's algorithm converting any regular e

Theorem: For any regu is an NFA N such that

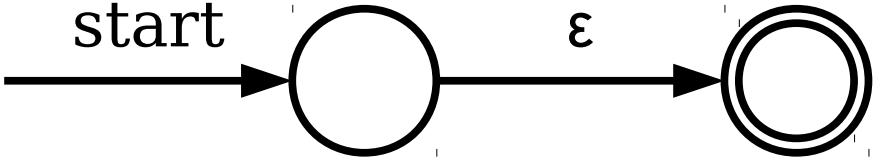
$$\mathscr{L}(R) = \mathscr{L}(N)$$

These are stronger requirements than are necessary for a normal NFA. We enforce these rules to simplify the construction.

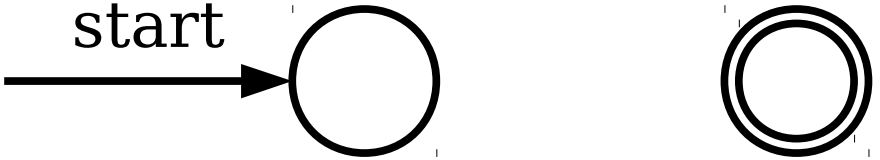
- N has exactly one accepting state.
- N has no transitions into its start state.
- N has no transitions out of its accepting state.



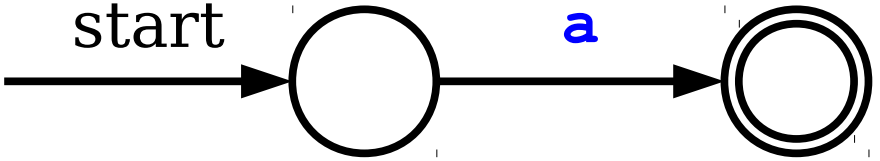
Base Cases



Automaton for ε



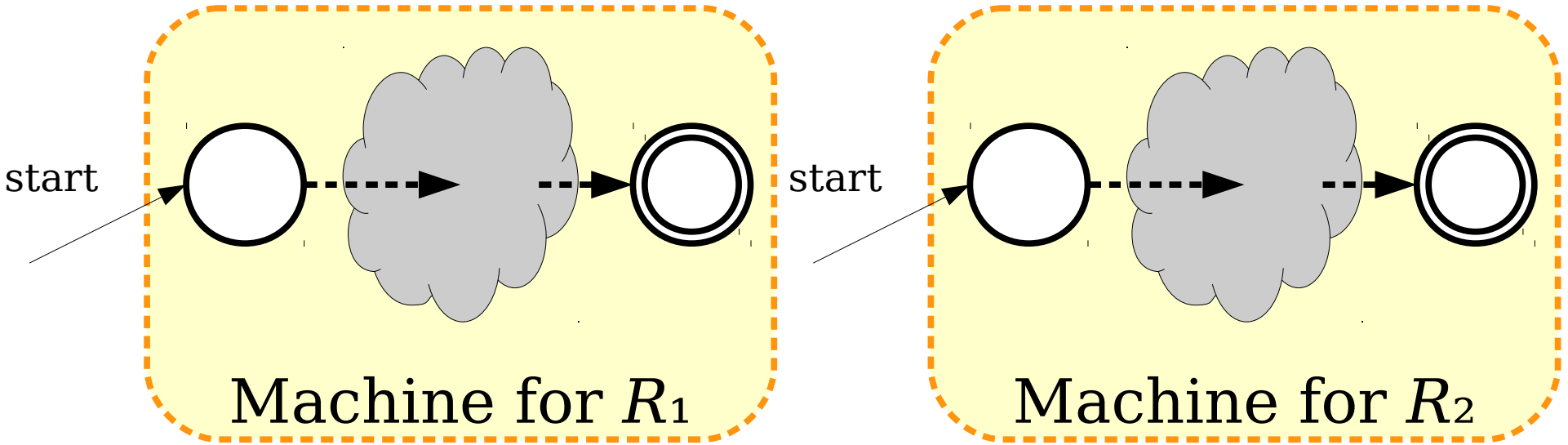
Automaton for Ø



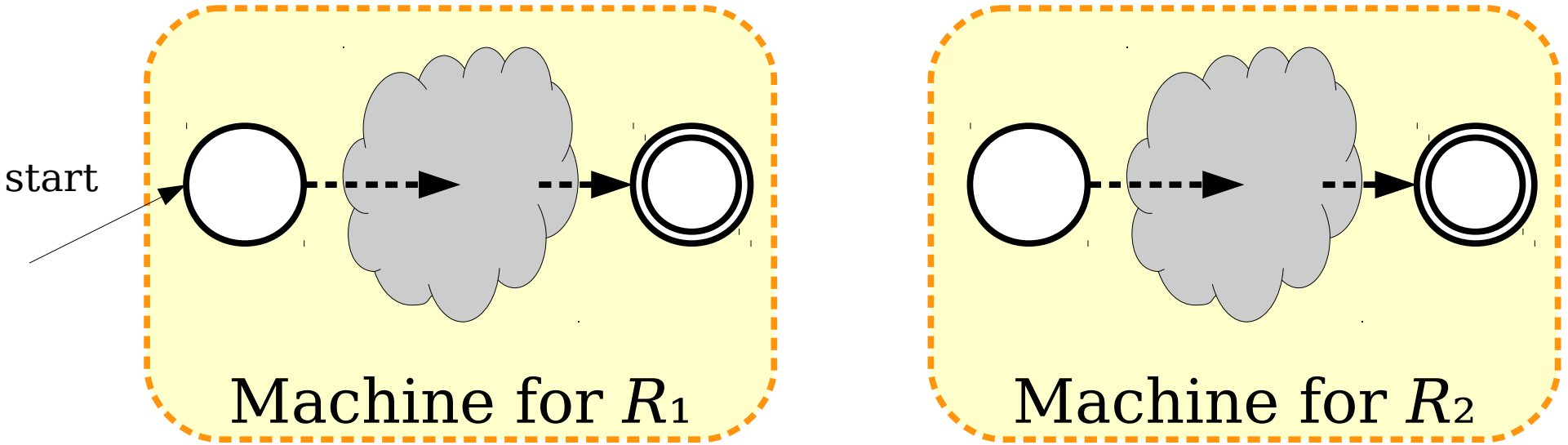
Automaton for single character **a**

Construction for $R_1 R_2$

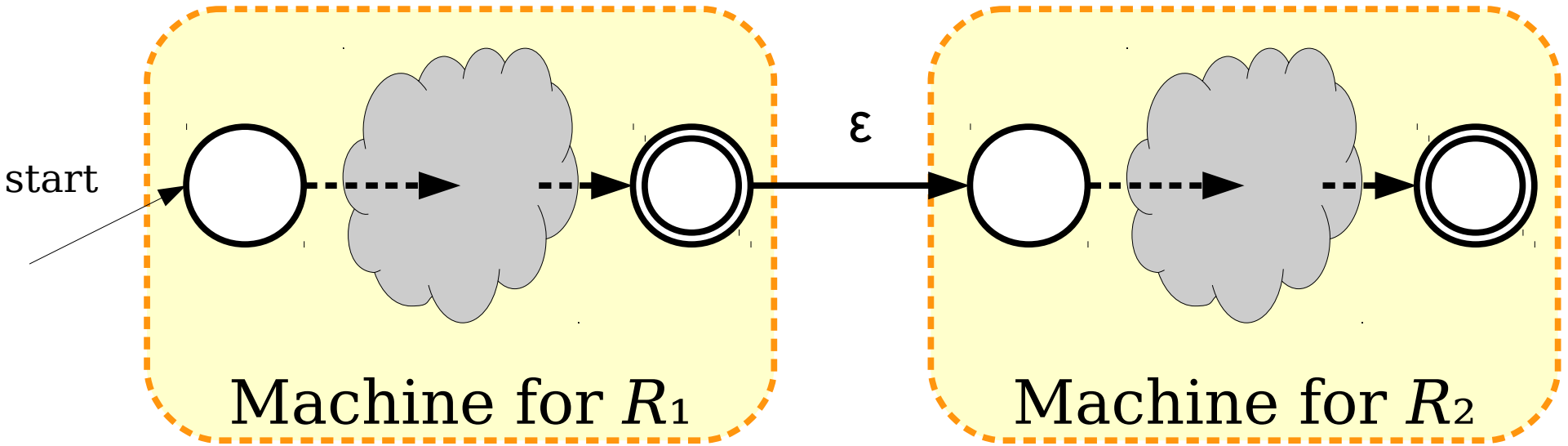
Construction for R_1R_2



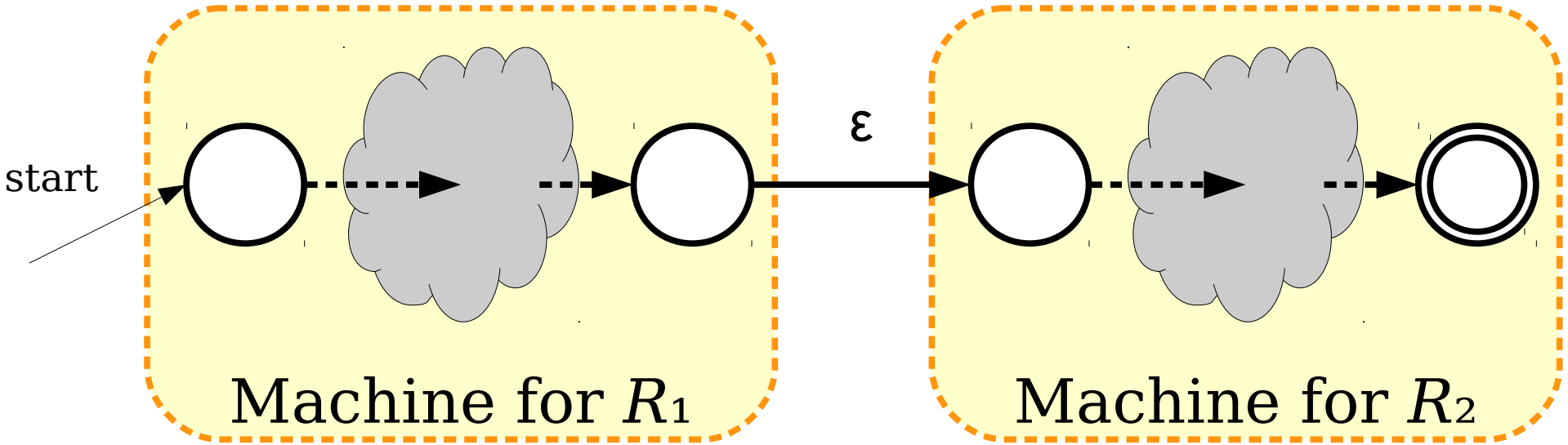
Construction for $R_1 R_2$



Construction for R_1R_2

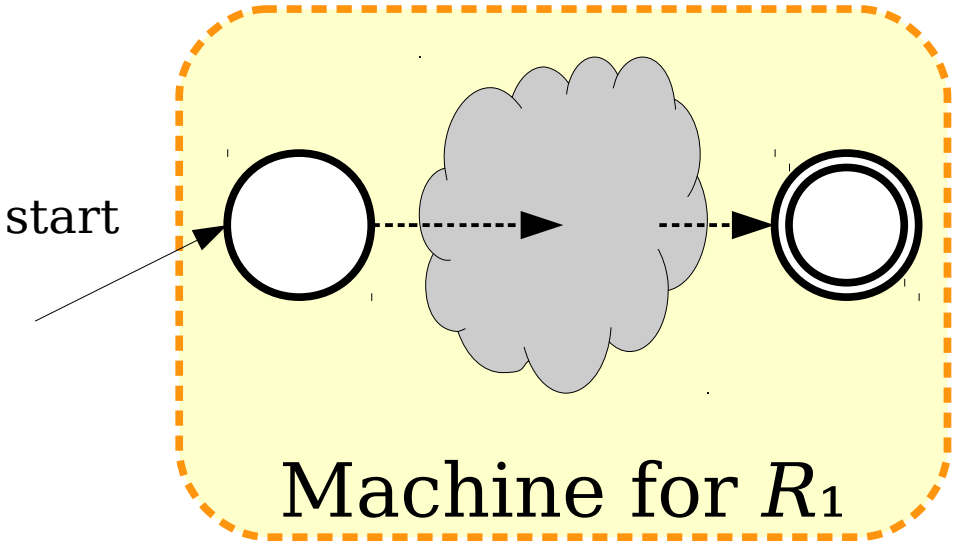


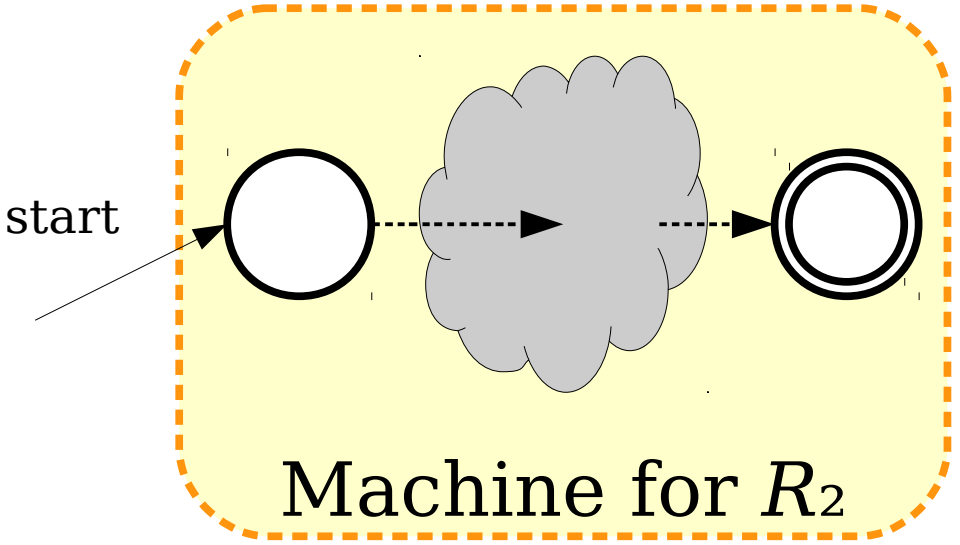
Construction for R_1R_2



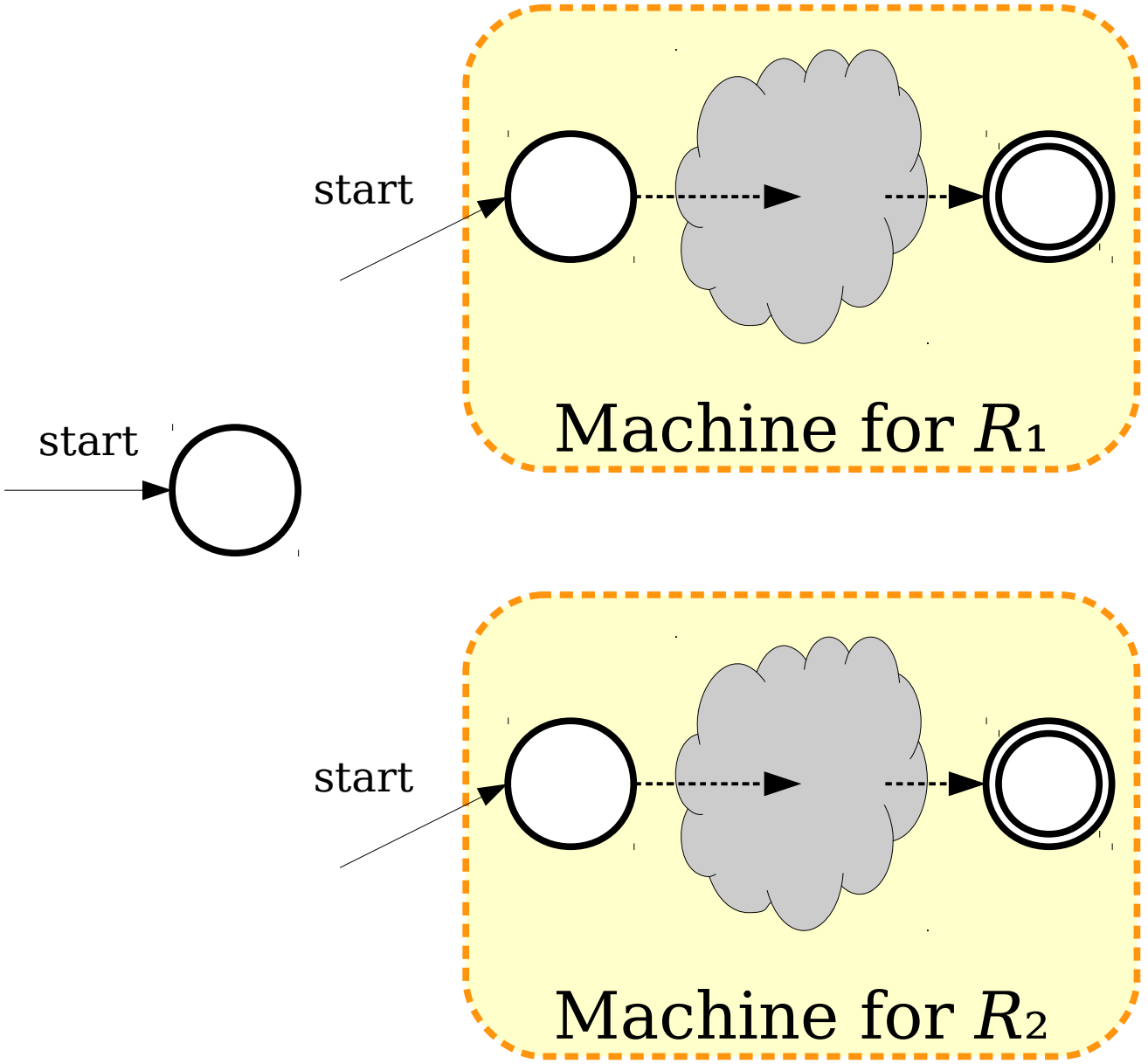
Construction for $R_1 \cup R_2$

Construction for $R_1 \cup R_2$

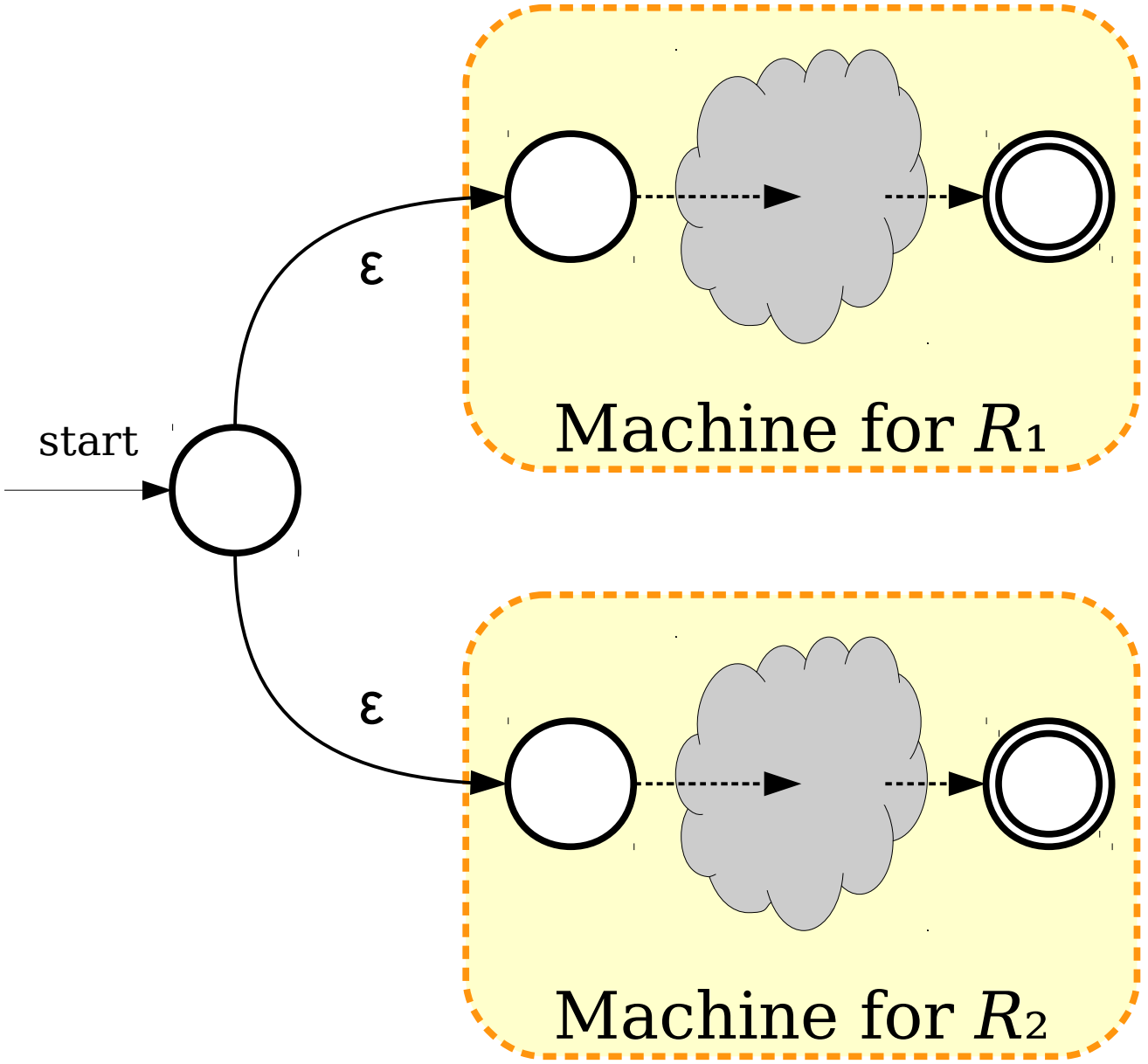




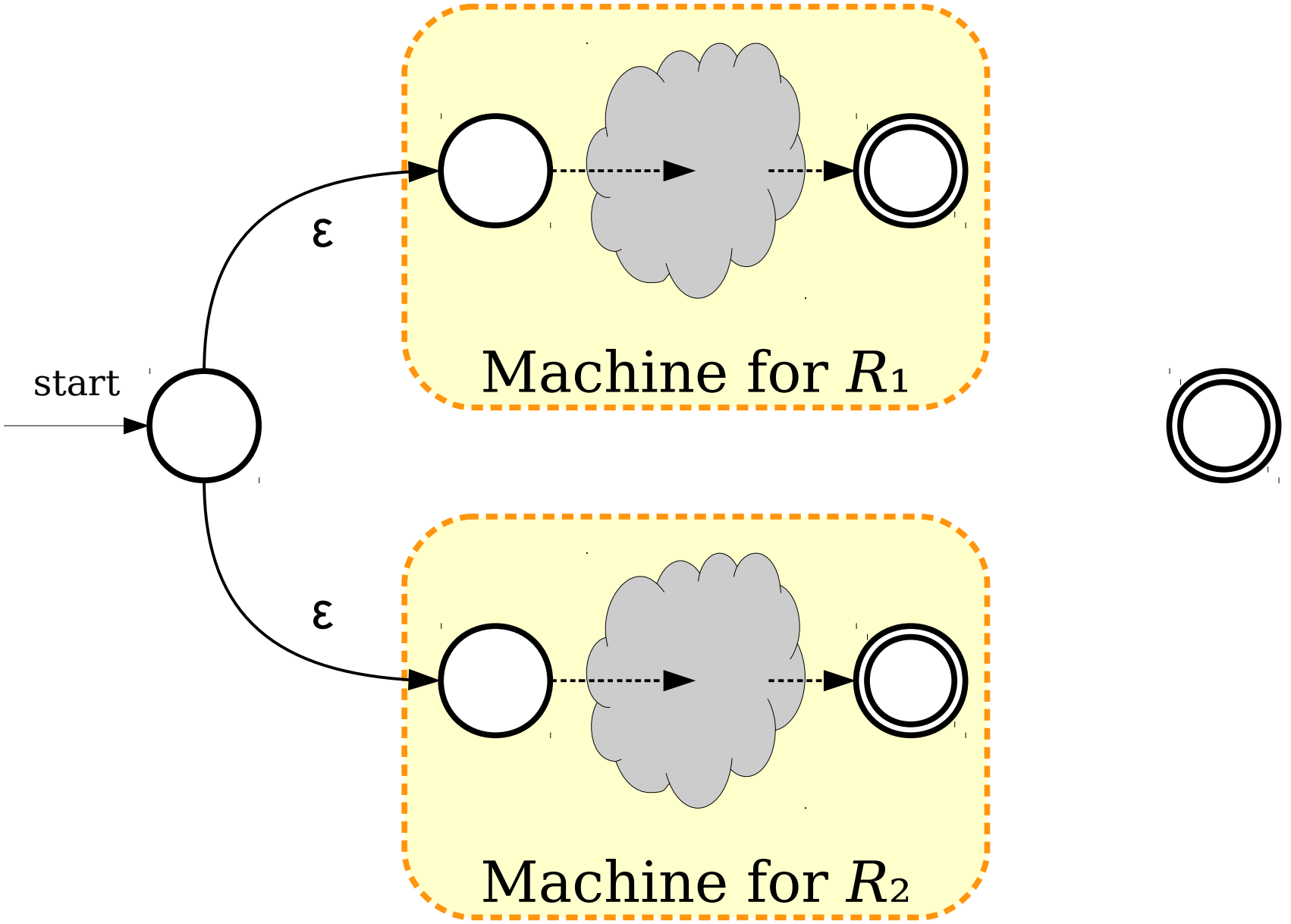
Construction for $R_1 \cup R_2$



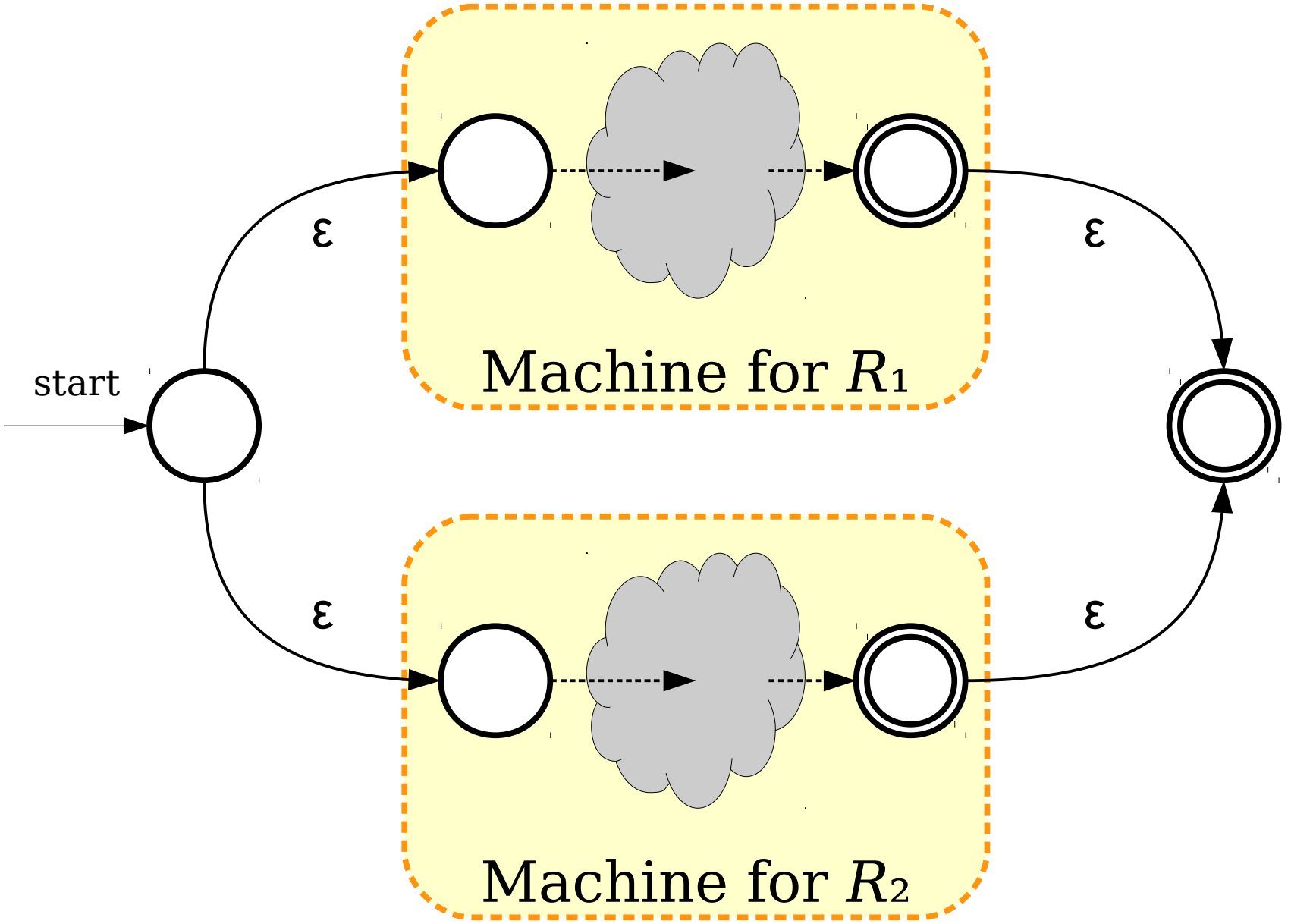
Construction for $R_1 \cup R_2$



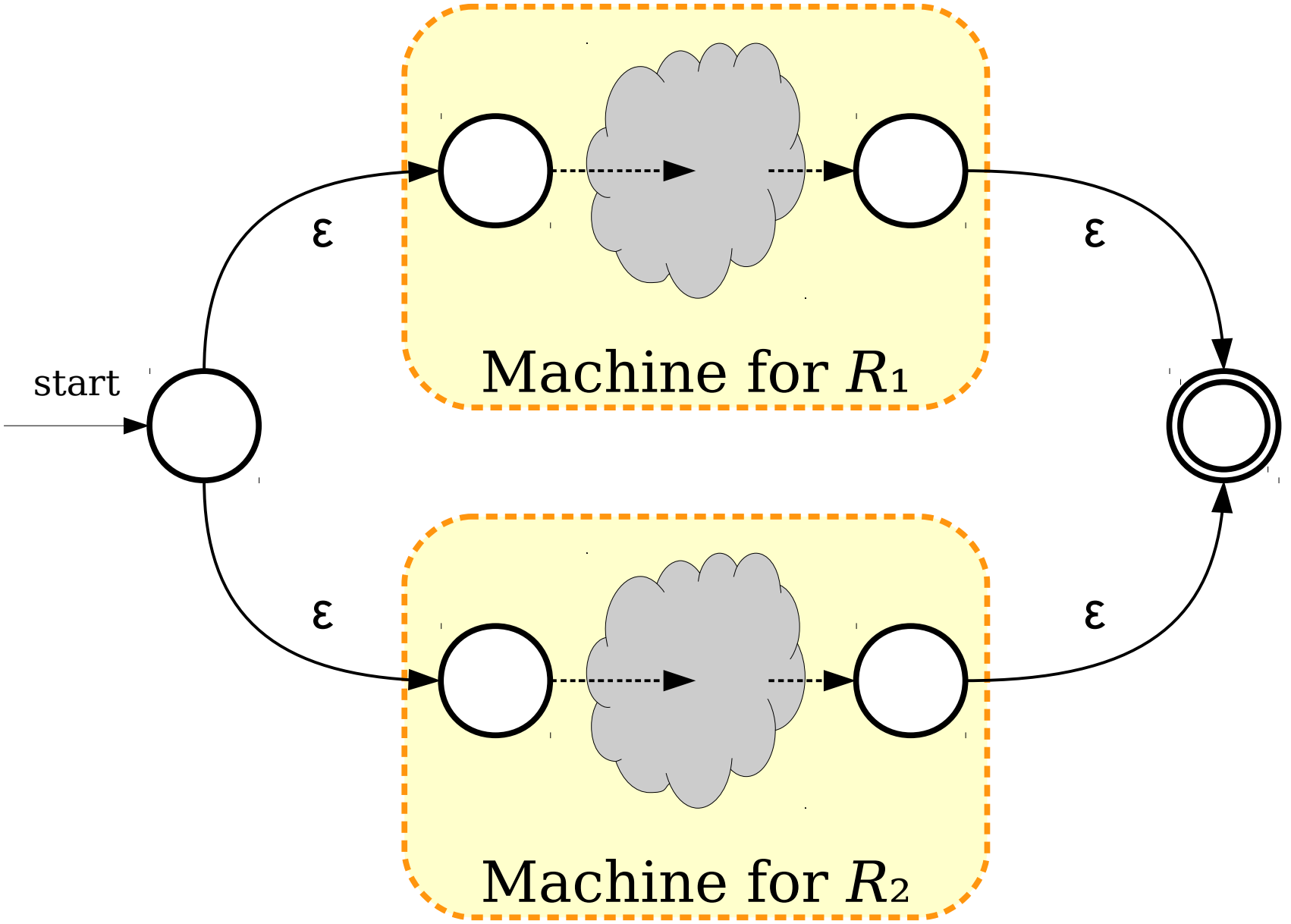
Construction for $R_1 \cup R_2$



Construction for $R_1 \cup R_2$

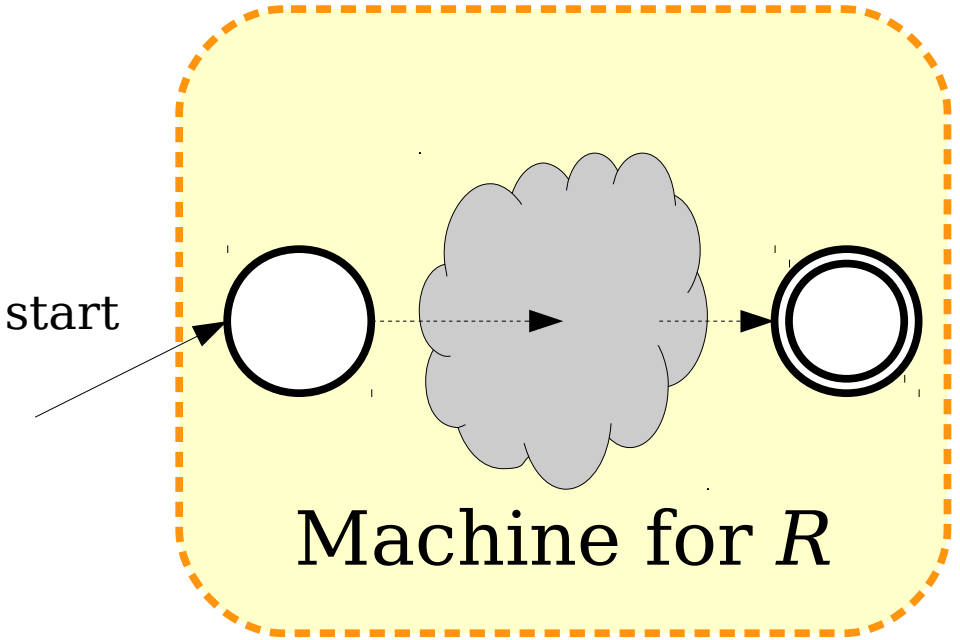


Construction for $R_1 \cup R_2$

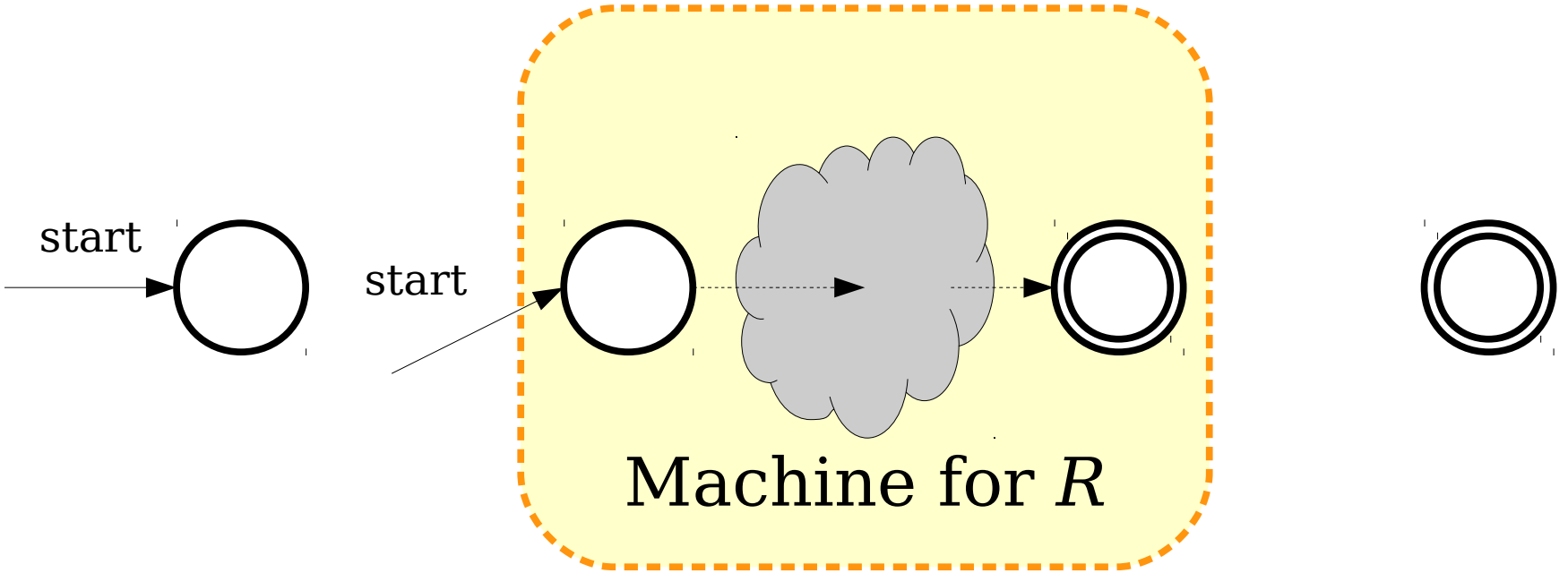


Construction for R^*

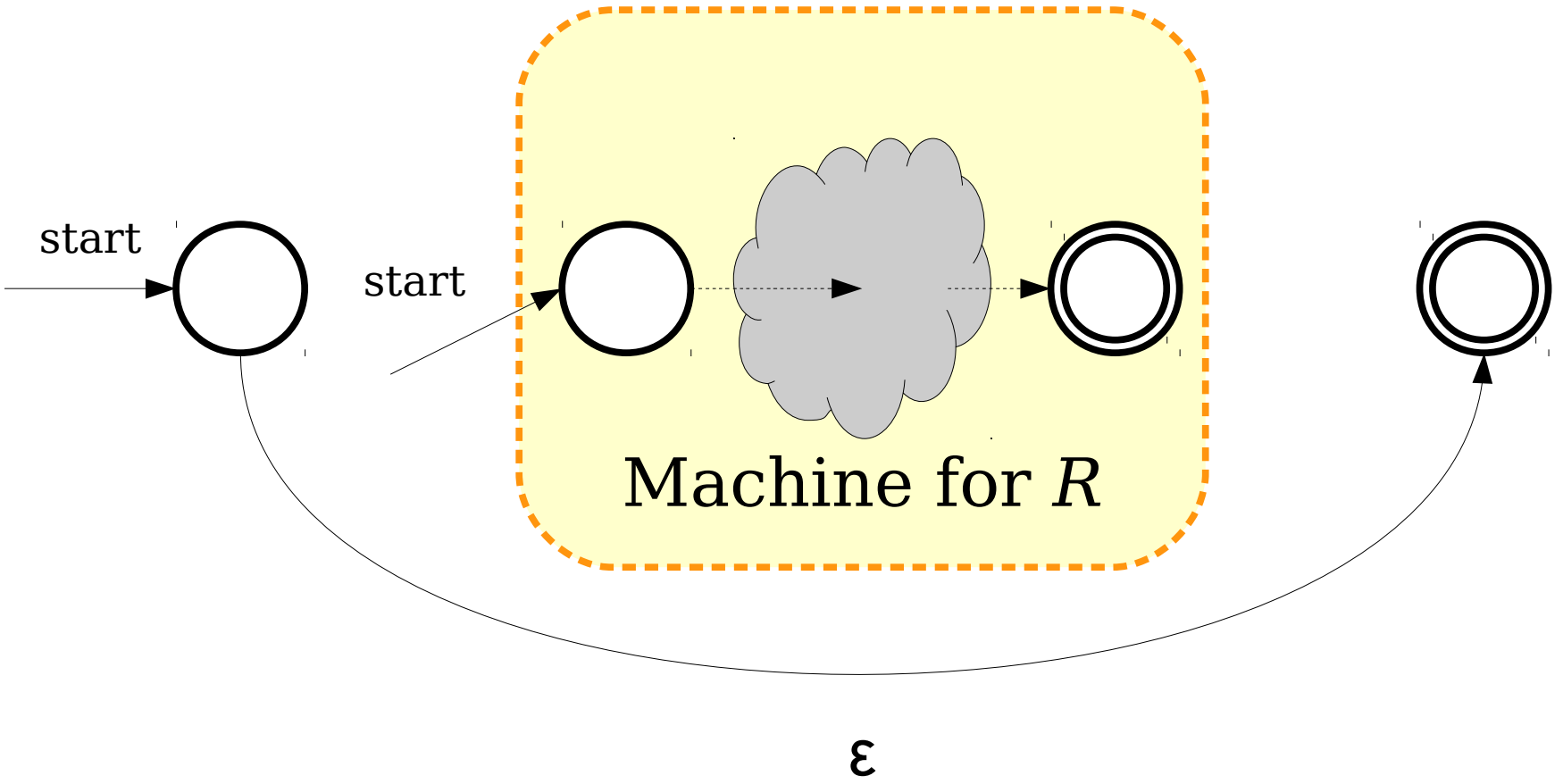
Construction for R^*



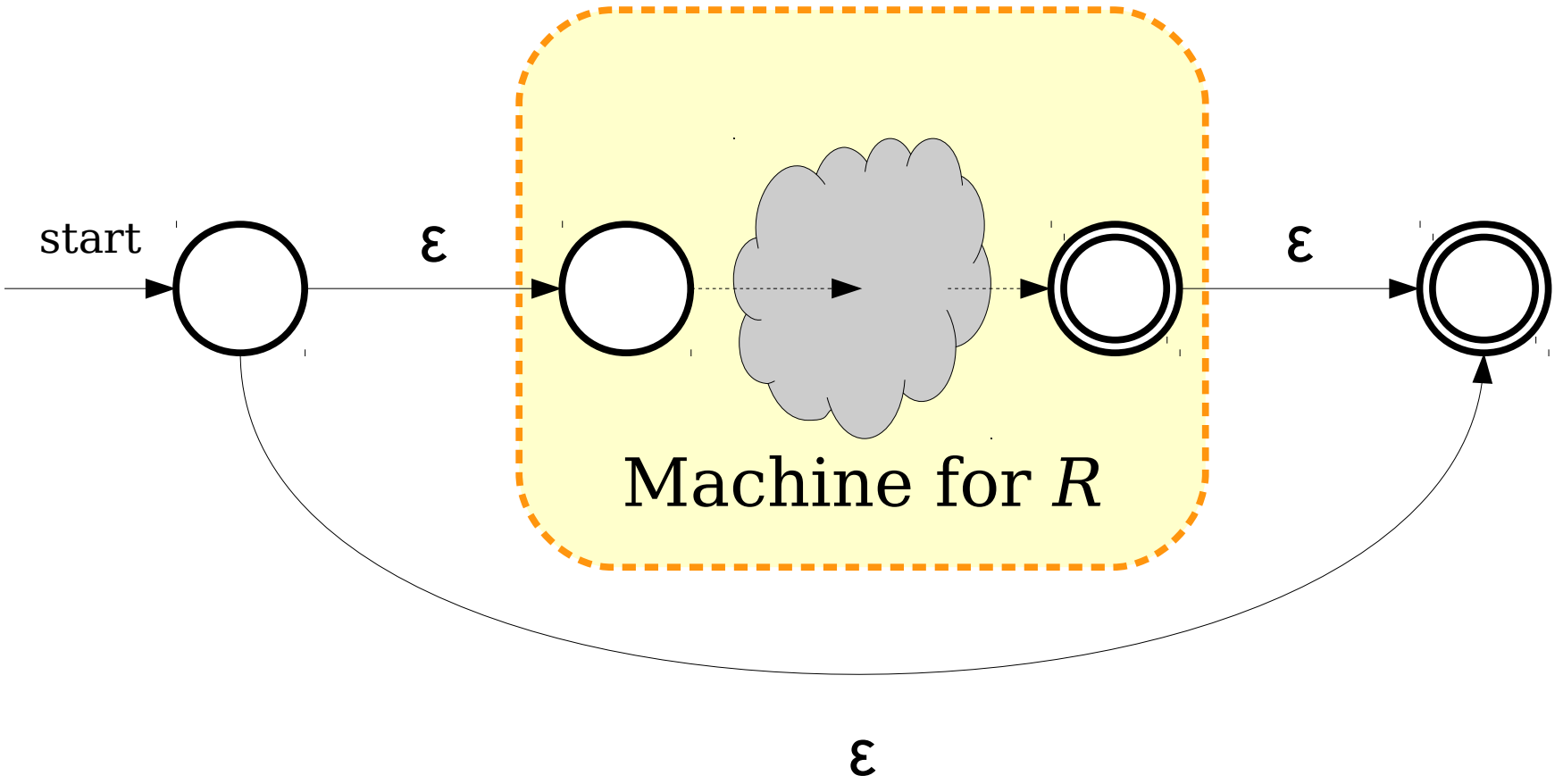
Construction for R^*



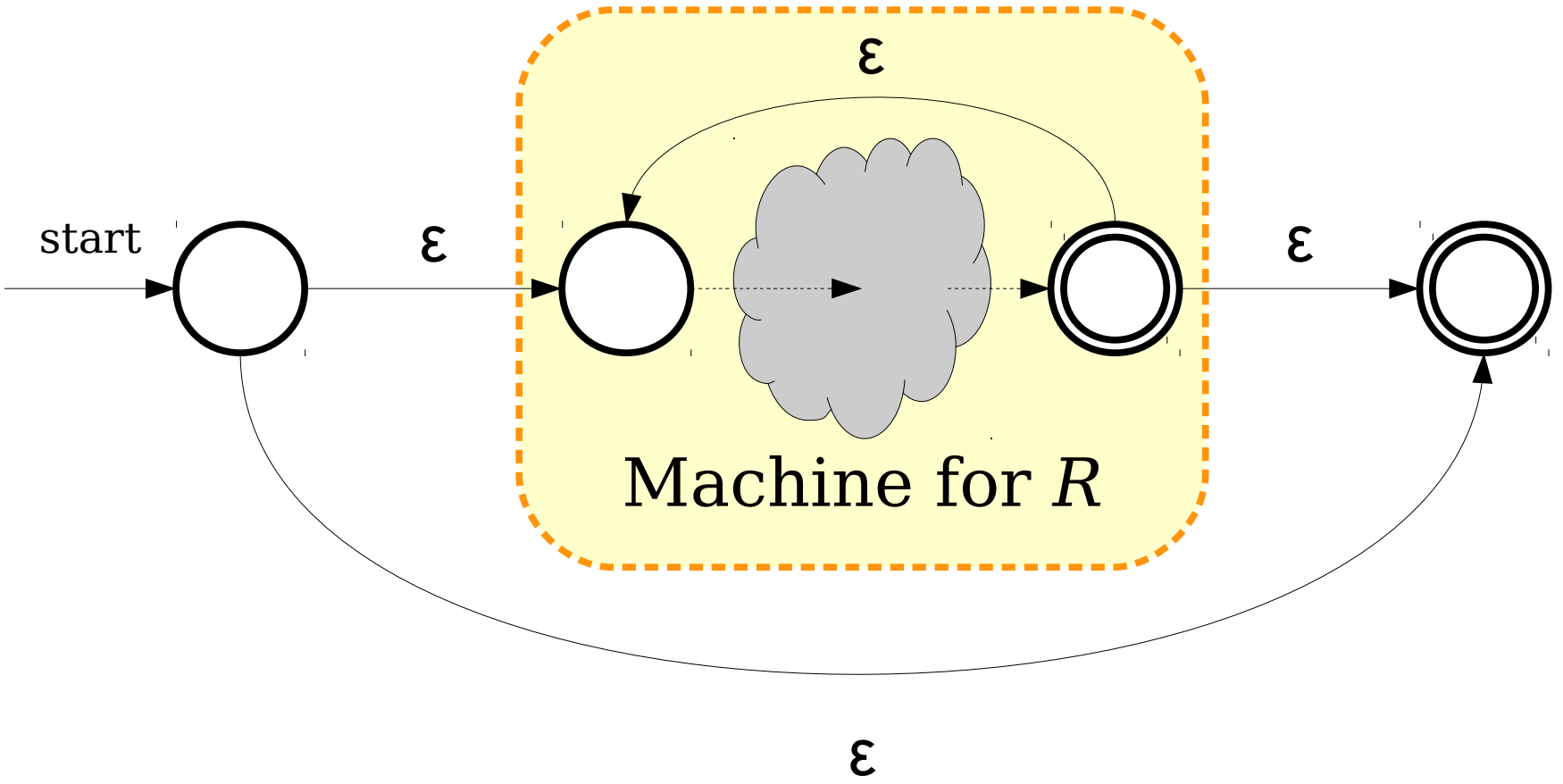
Construction for R^*



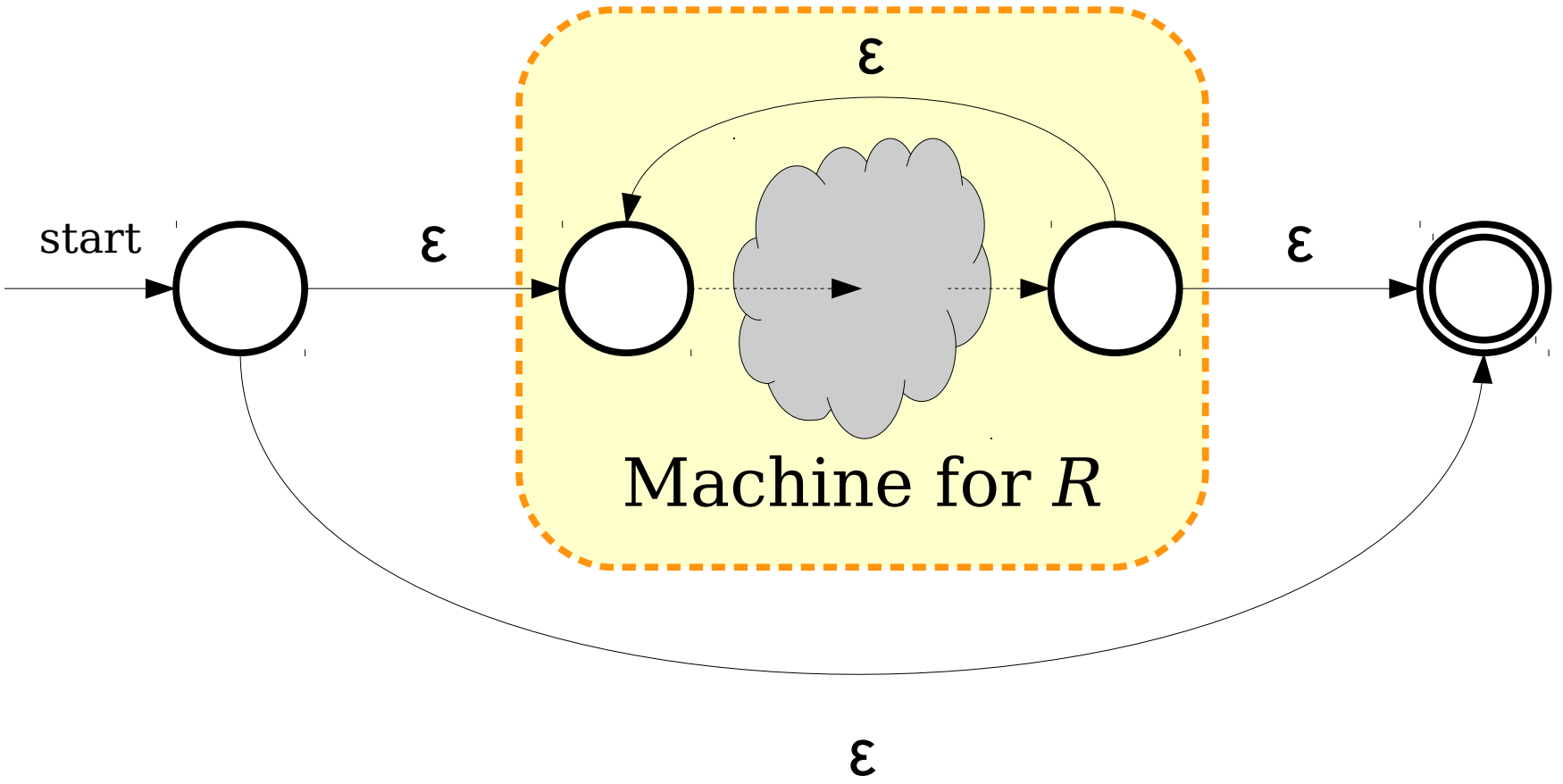
Construction for R^*



Construction for R^*



Construction for R^*



Why This Matters

- Many software tools work by matching regular expressions against text.
- One possible algorithm for doing so:
  - Convert the regular expression to an NFA.
  - (Optionally) Convert the NFA to a DFA using the subset construction.
  - Run the text through the finite automaton and look for matches.
- This is actually used in practice! The compiled matching automata run extremely quickly.

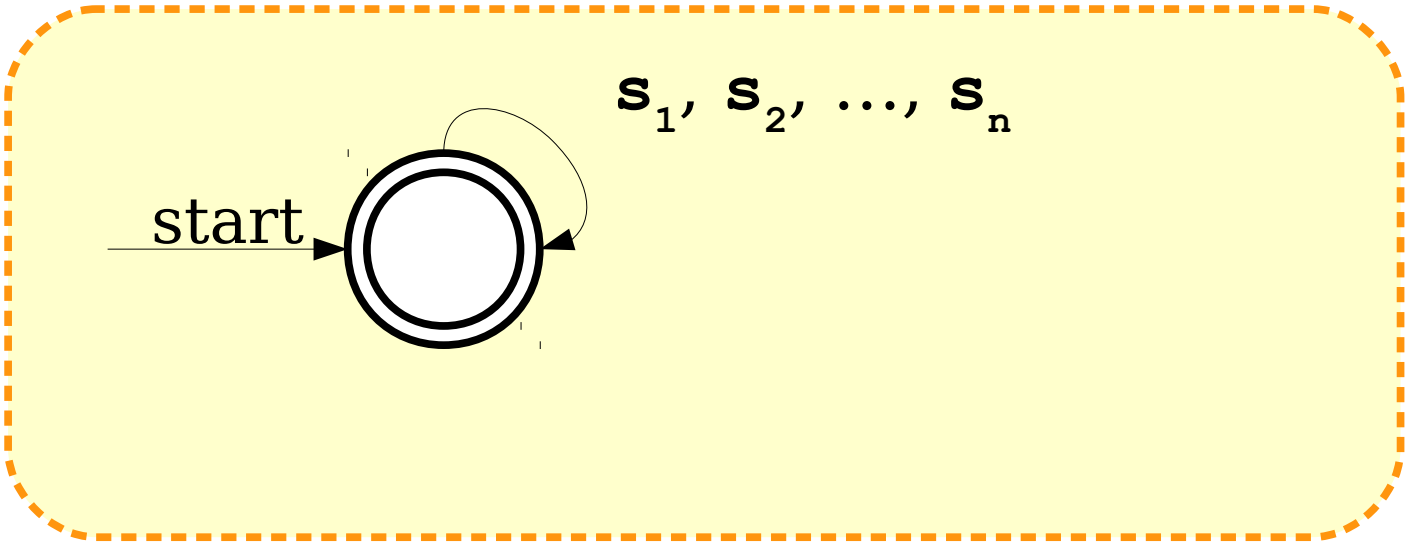
The Power of Regular Expressions

Theorem: If L is a regular language, then there is a regular expression for L.

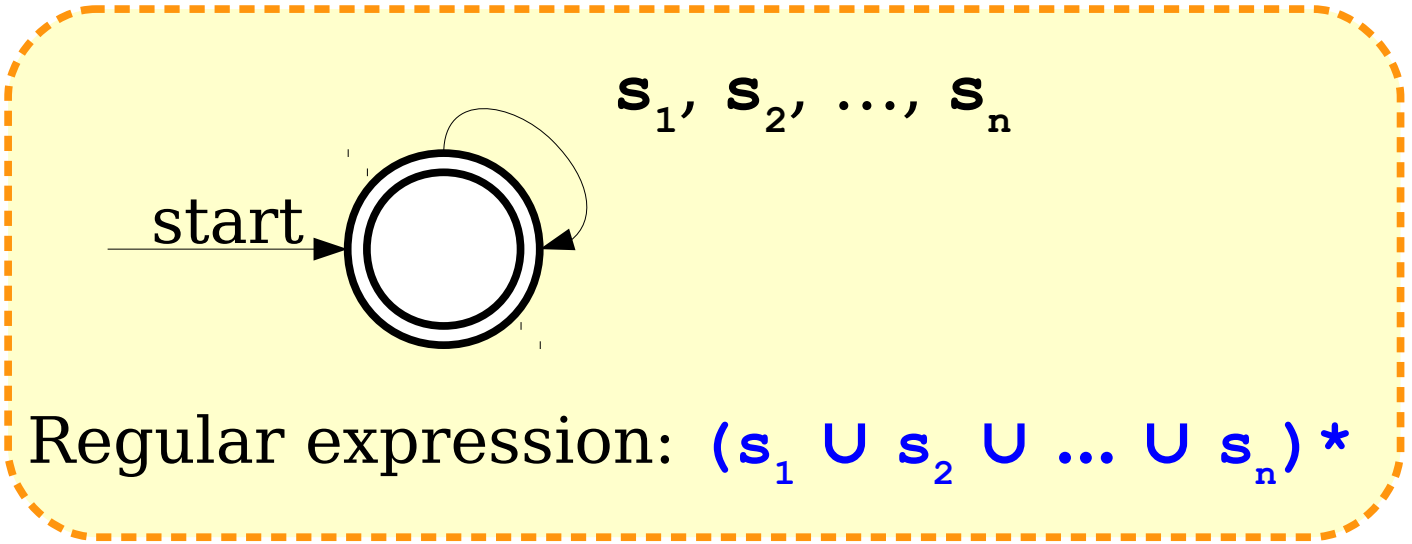
This is not obvious!

Proof idea: Show how to convert an arbitrary NFA into a regular expression.

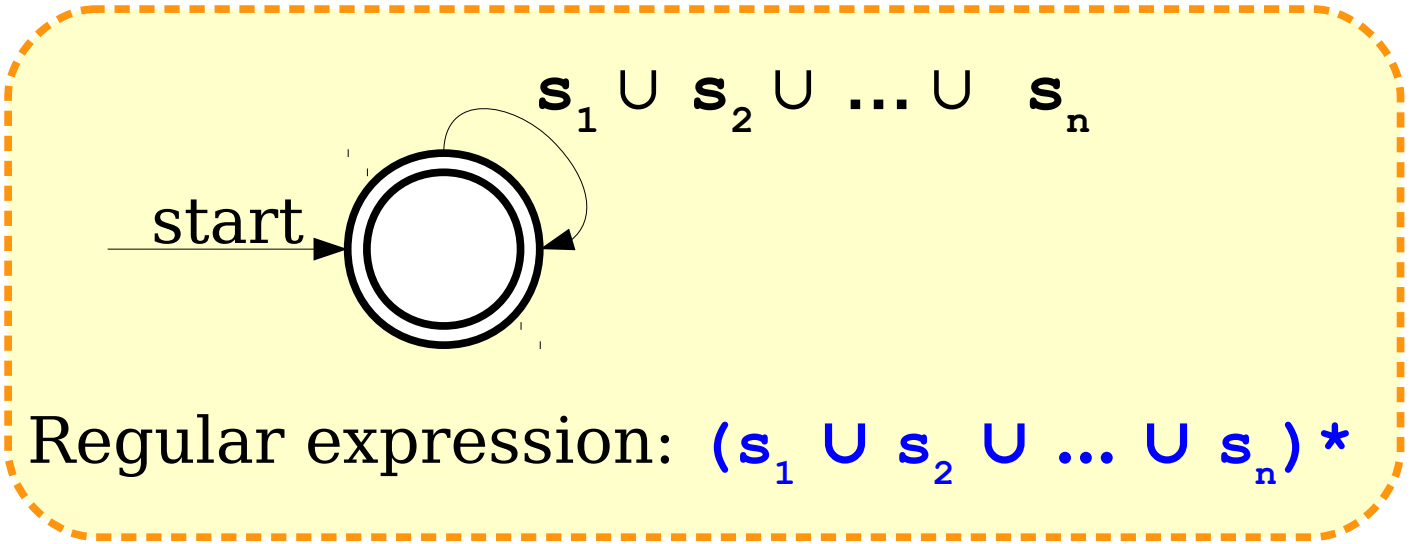
From NFAs to Regular Expressions



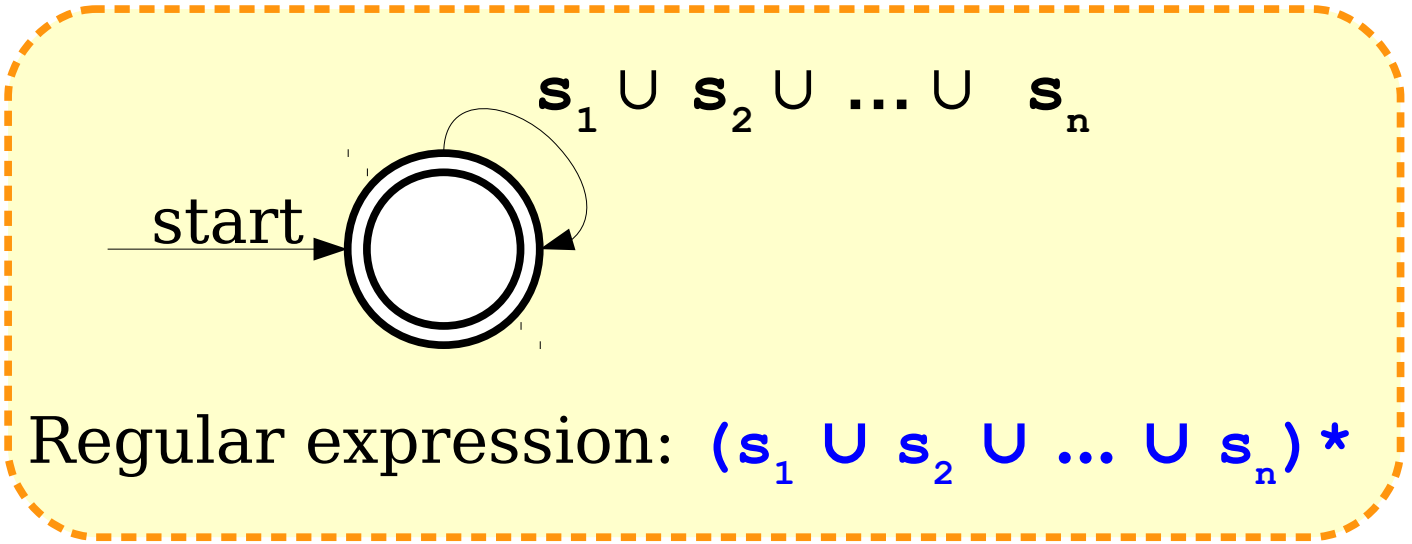
From NFAs to Regular Expressions



From NFAs to Regular Expressions



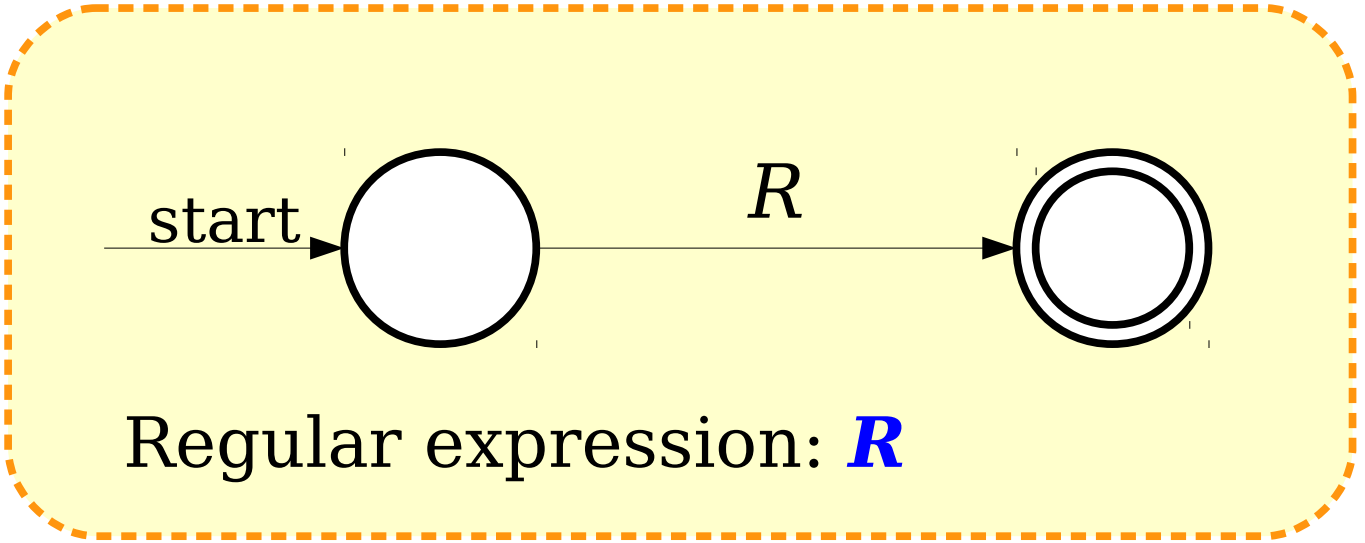
From NFAs to Regular Expressions



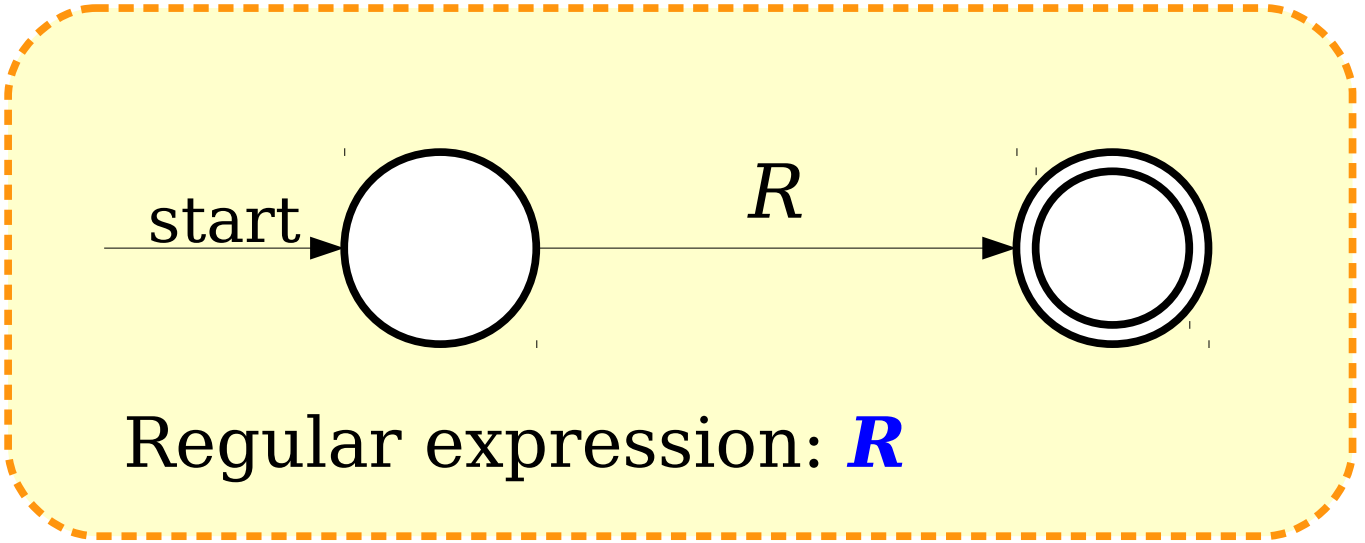
Key idea: Label transitions with arbitrary regular expressions.

From NFAs to Regular Expressions

From NFAs to Regular Expressions

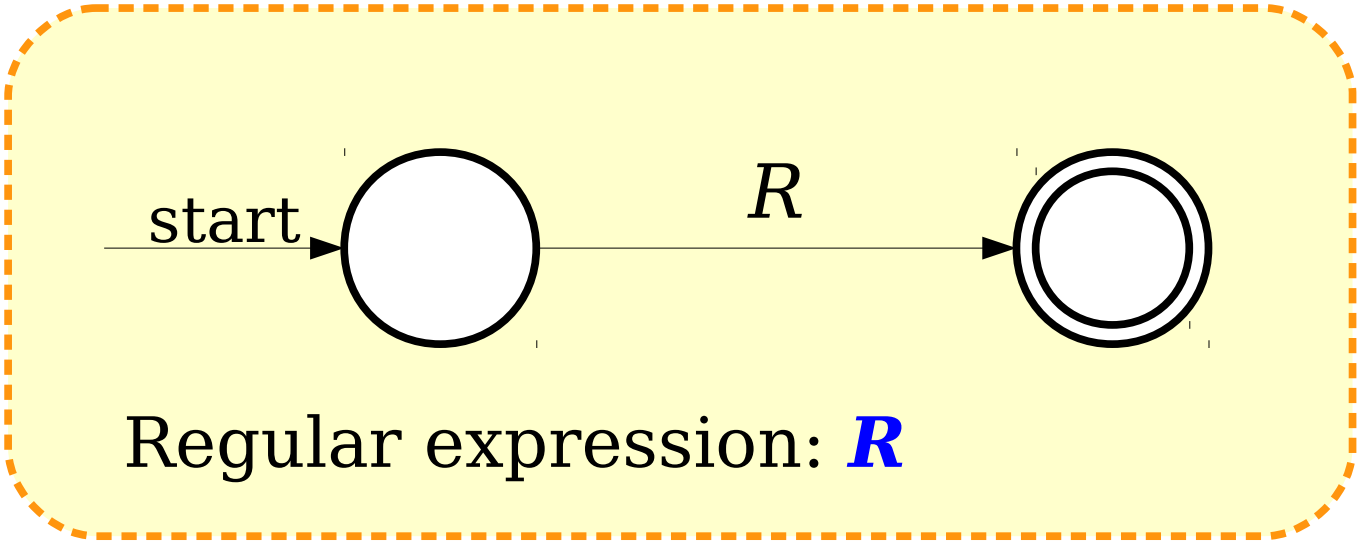


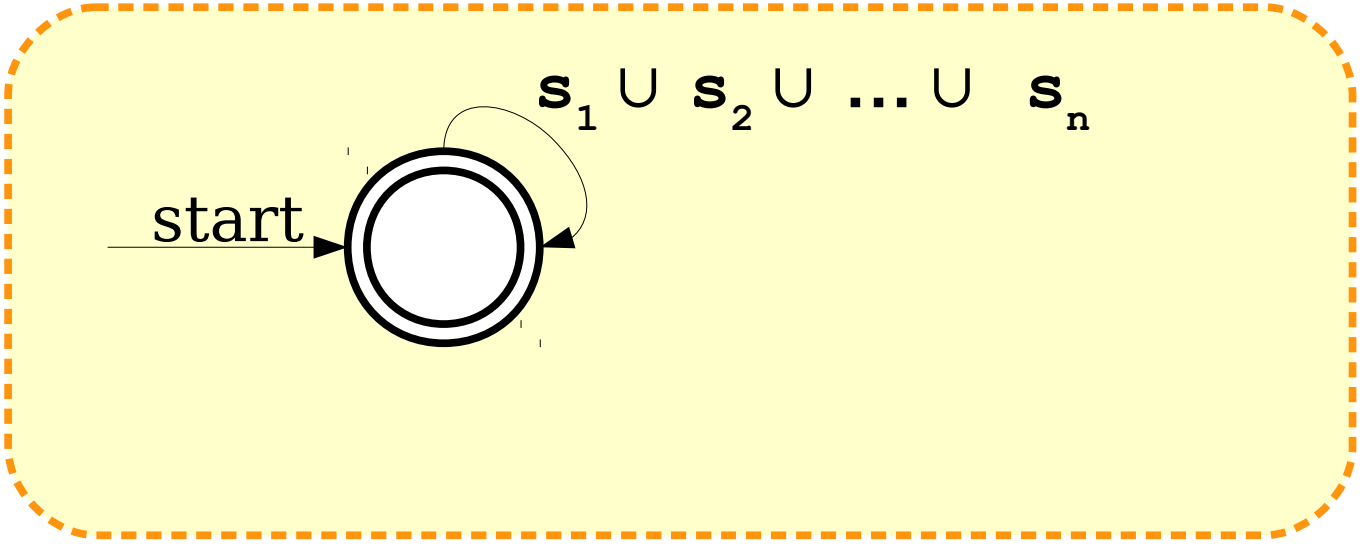
From NFAs to Regular Expressions



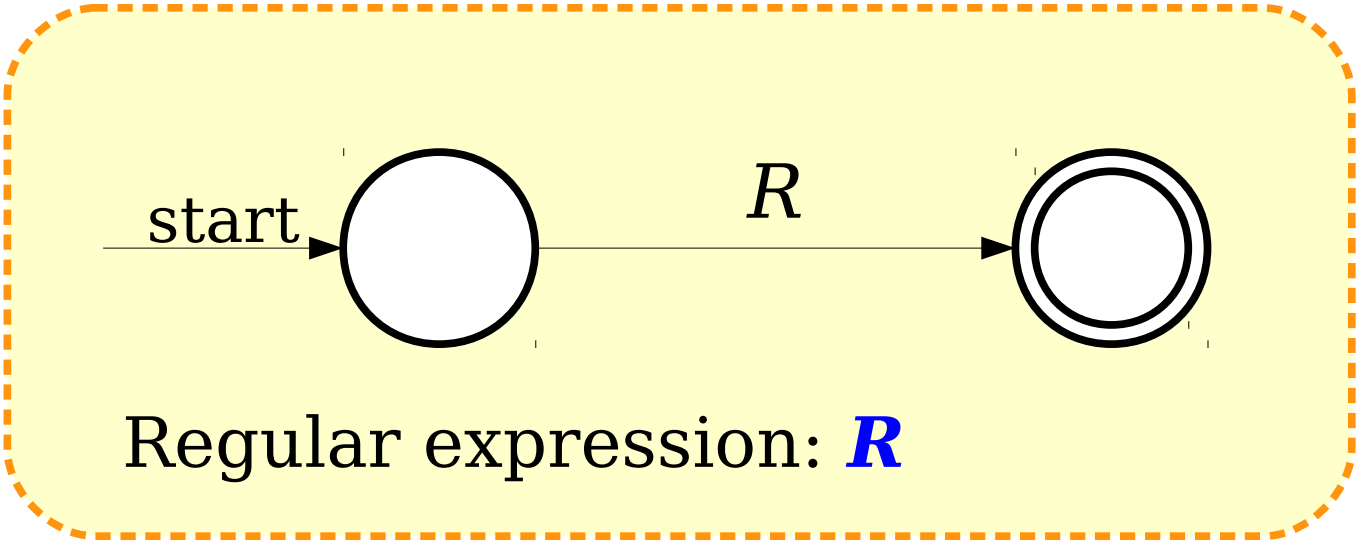
Key idea: If we can convert any NFA into something that looks like this, we can easily read off the regular expression.

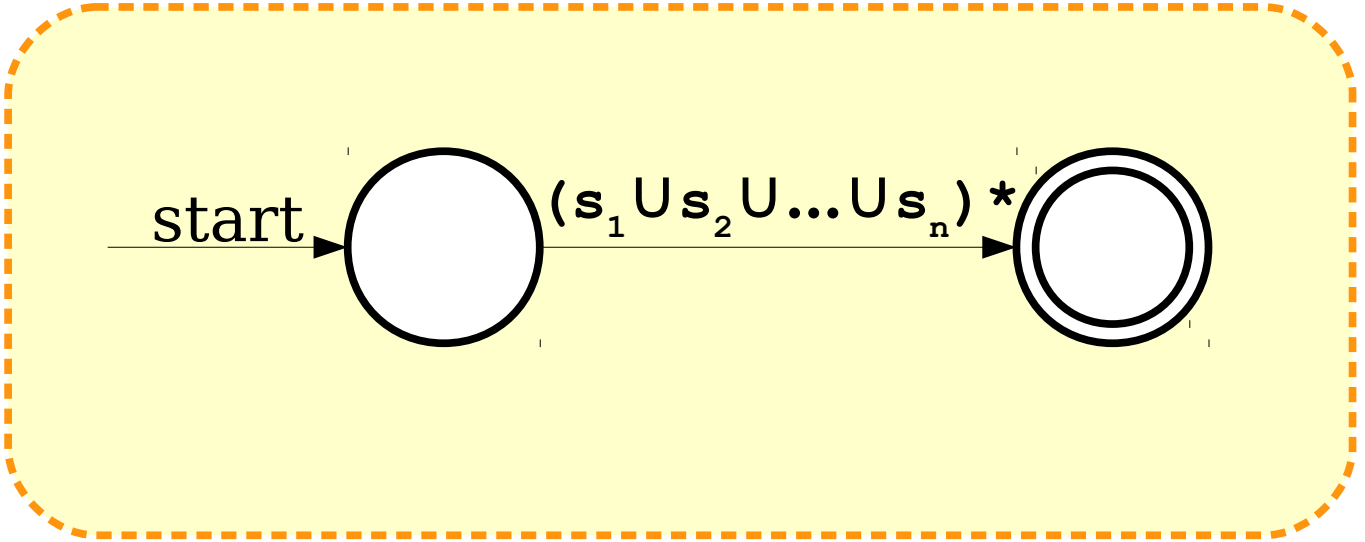
From NFAs to Regular Expressions



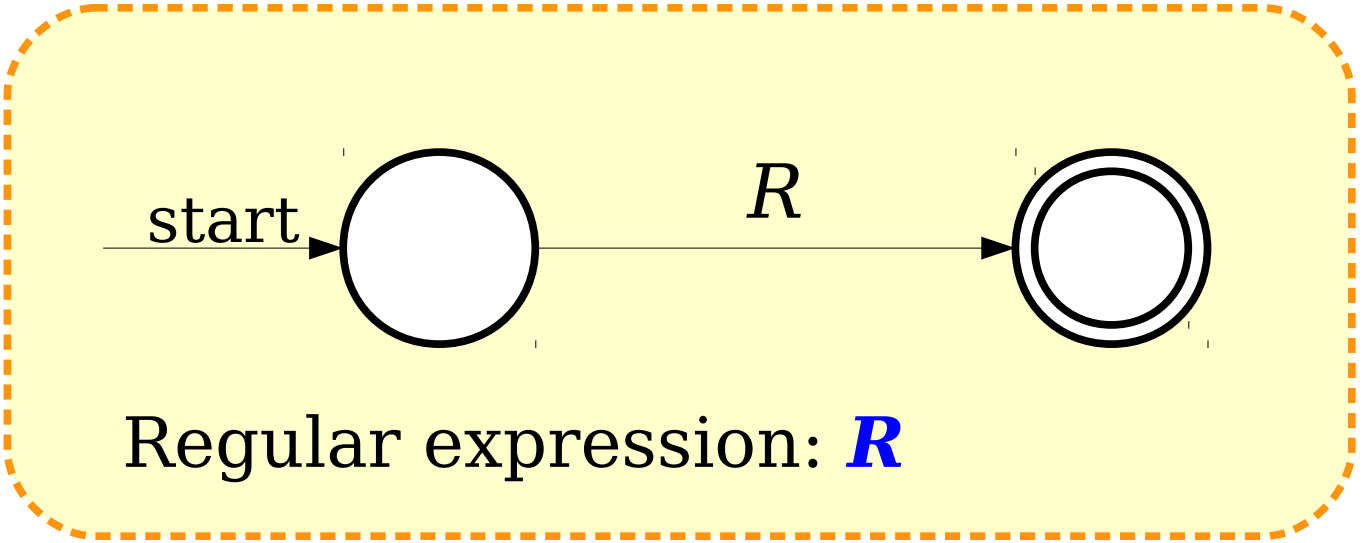


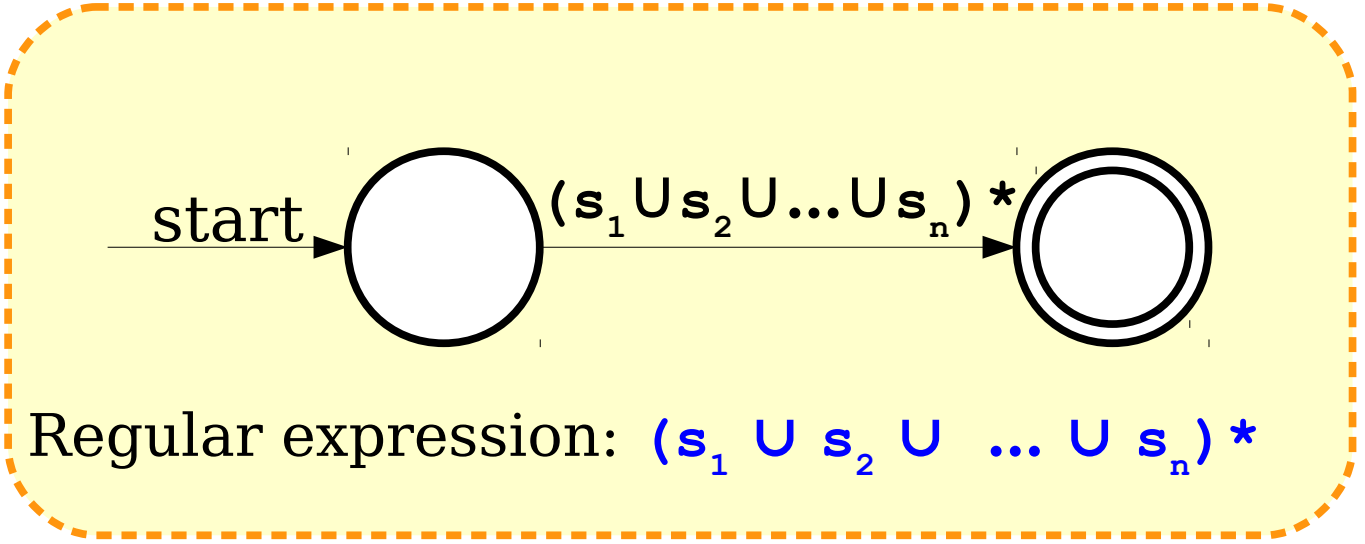
From NFAs to Regular Expressions



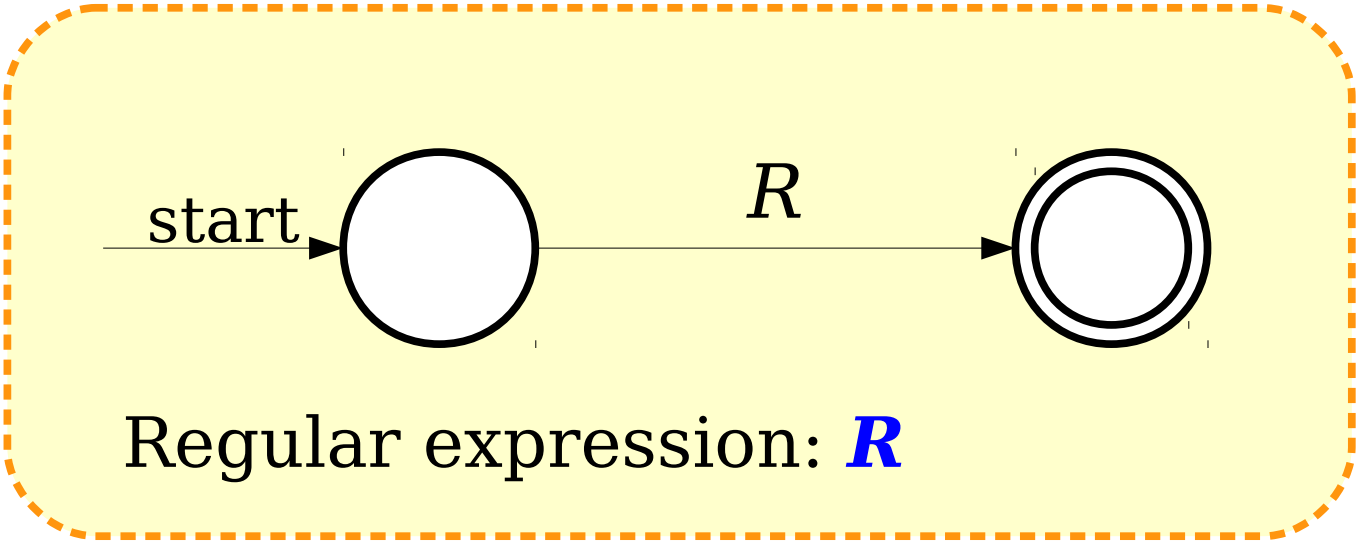


From NFAs to Regular Expressions

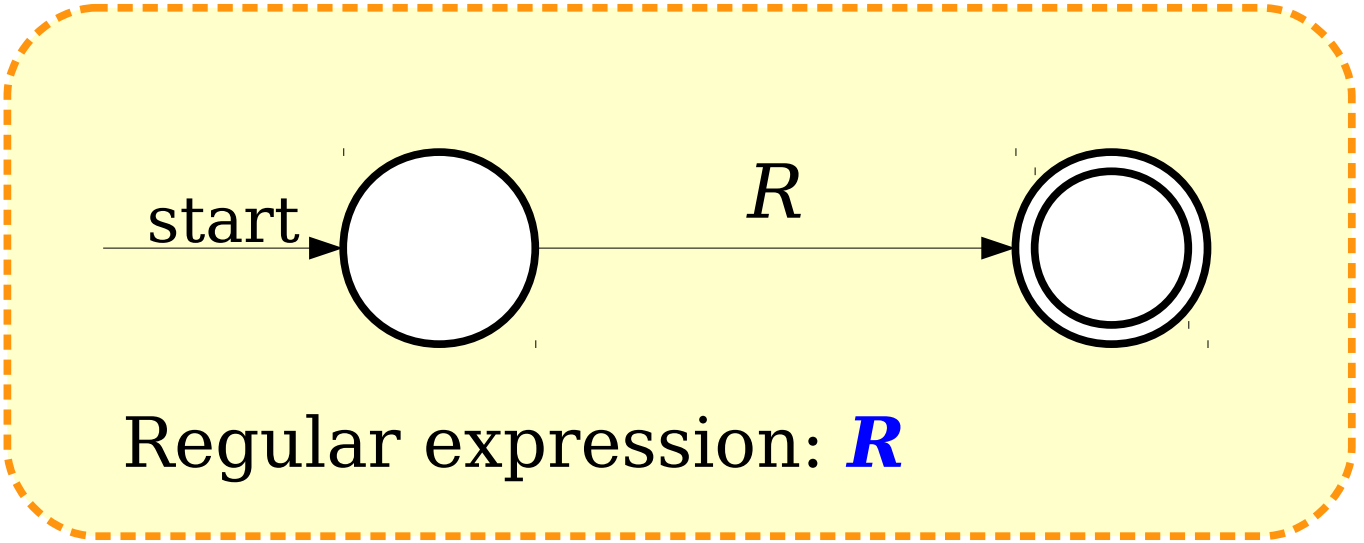


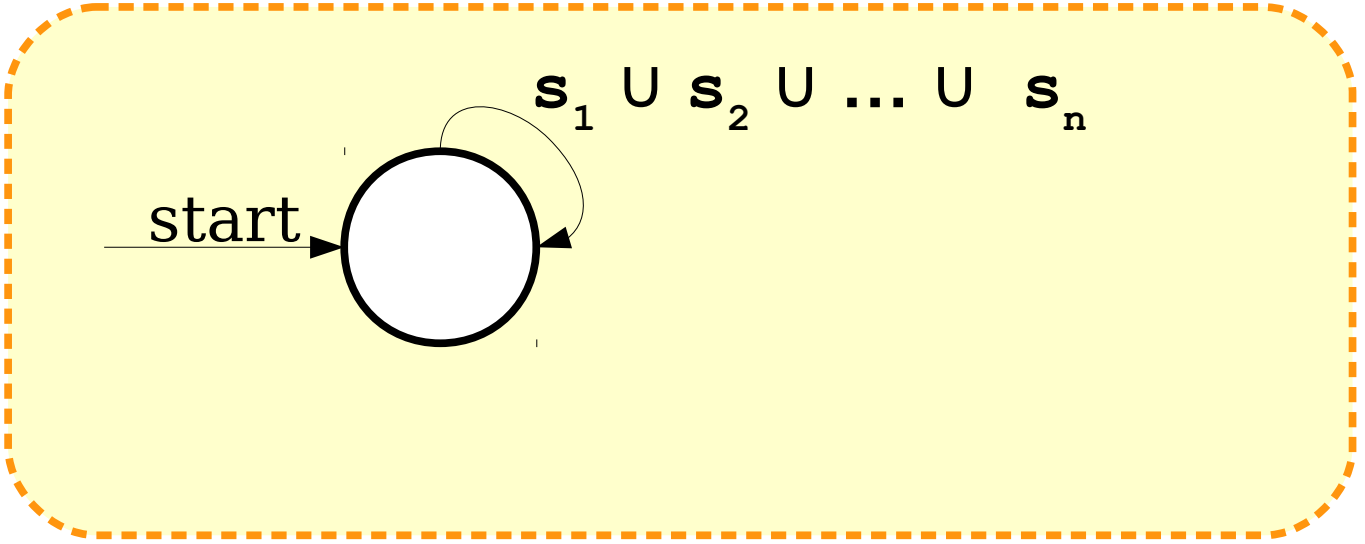


From NFAs to Regular Expressions

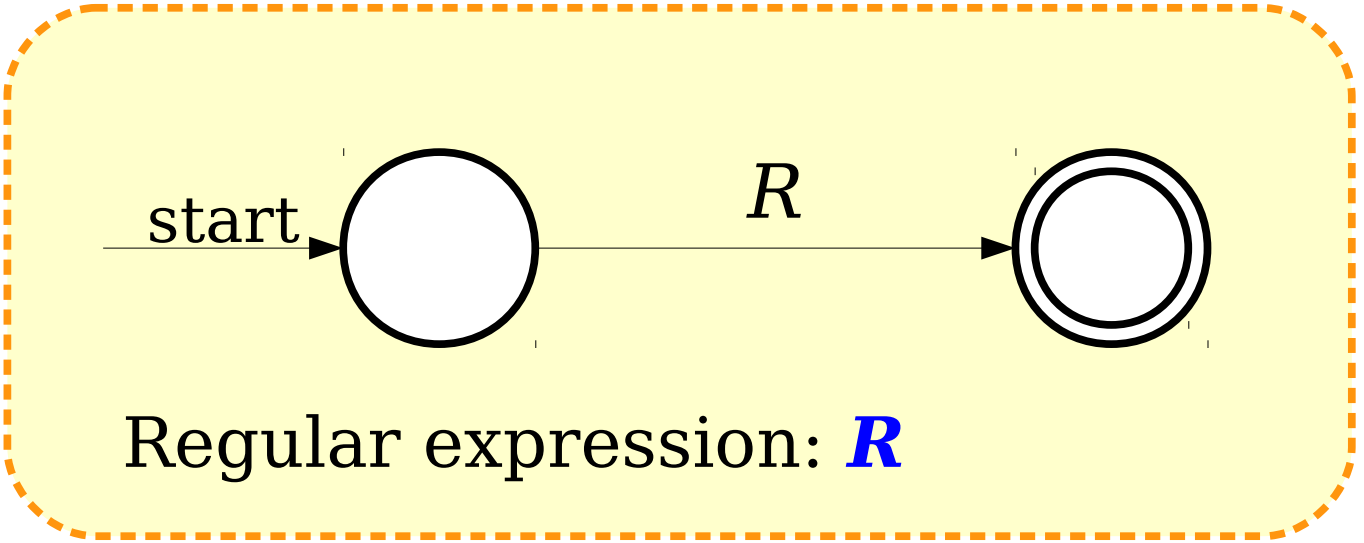


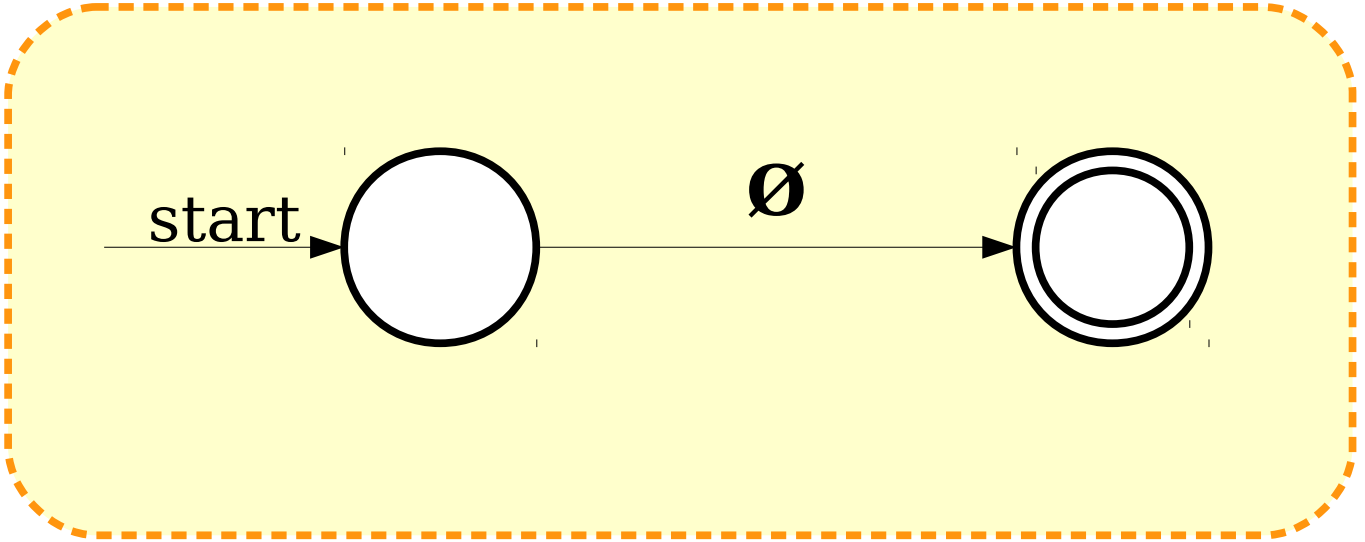
From NFAs to Regular Expressions



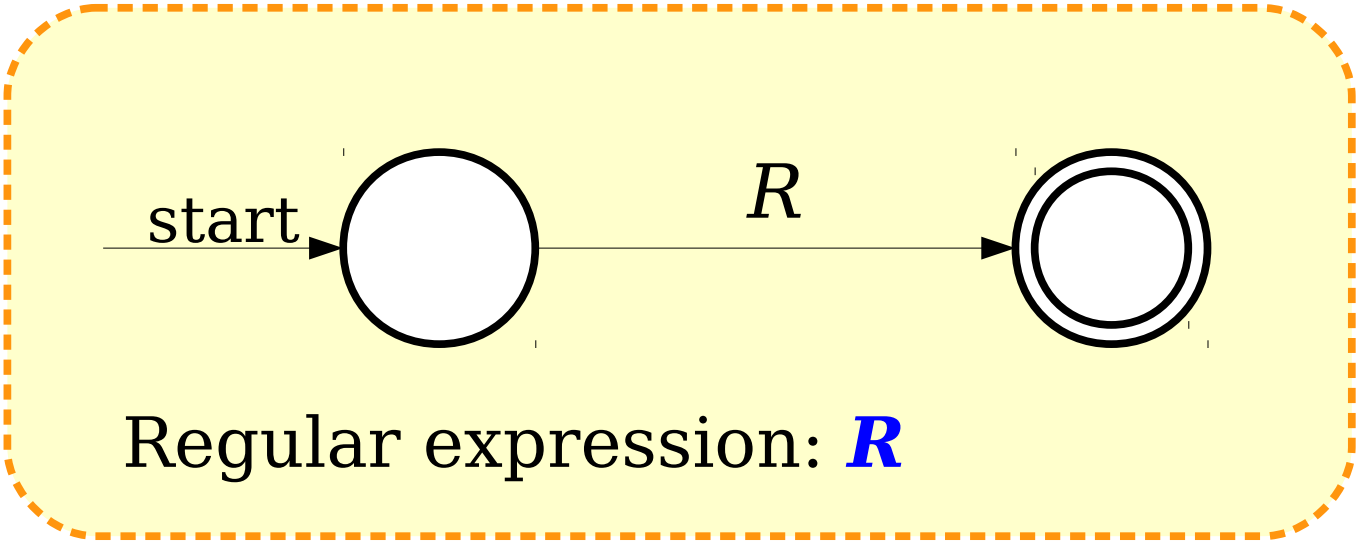


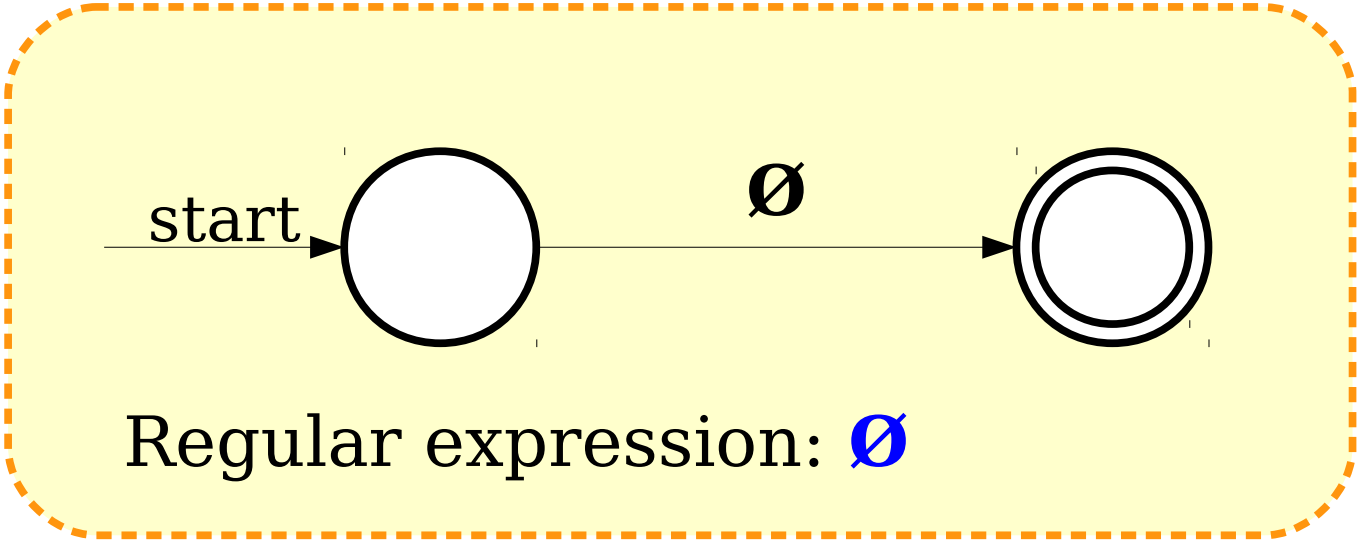
From NFAs to Regular Expressions



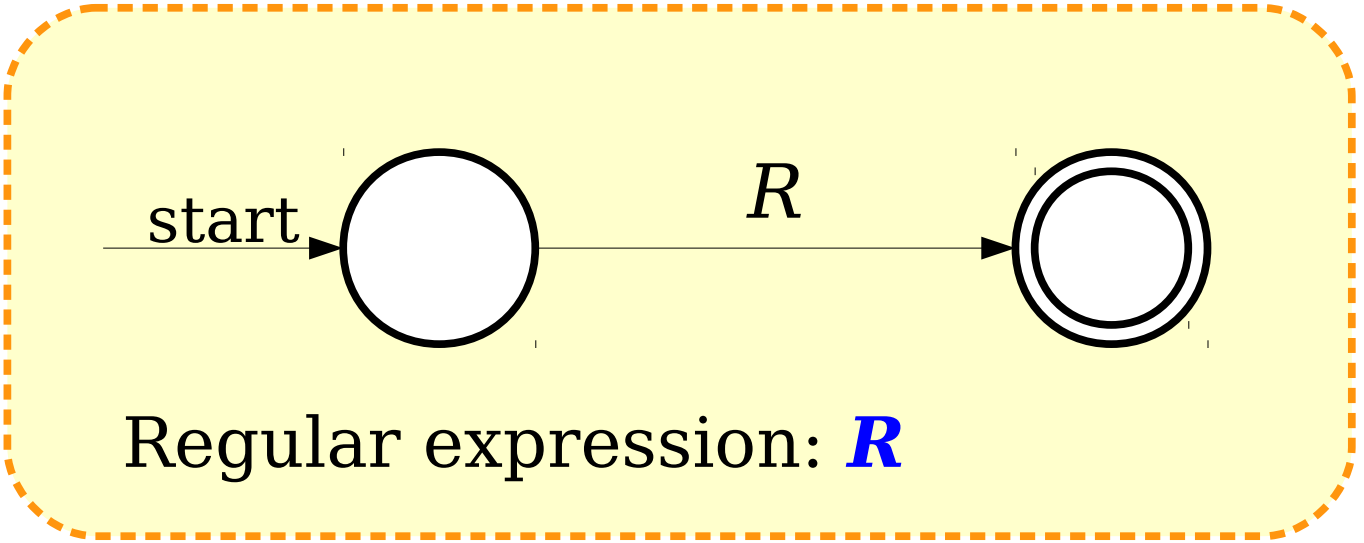


From NFAs to Regular Expressions

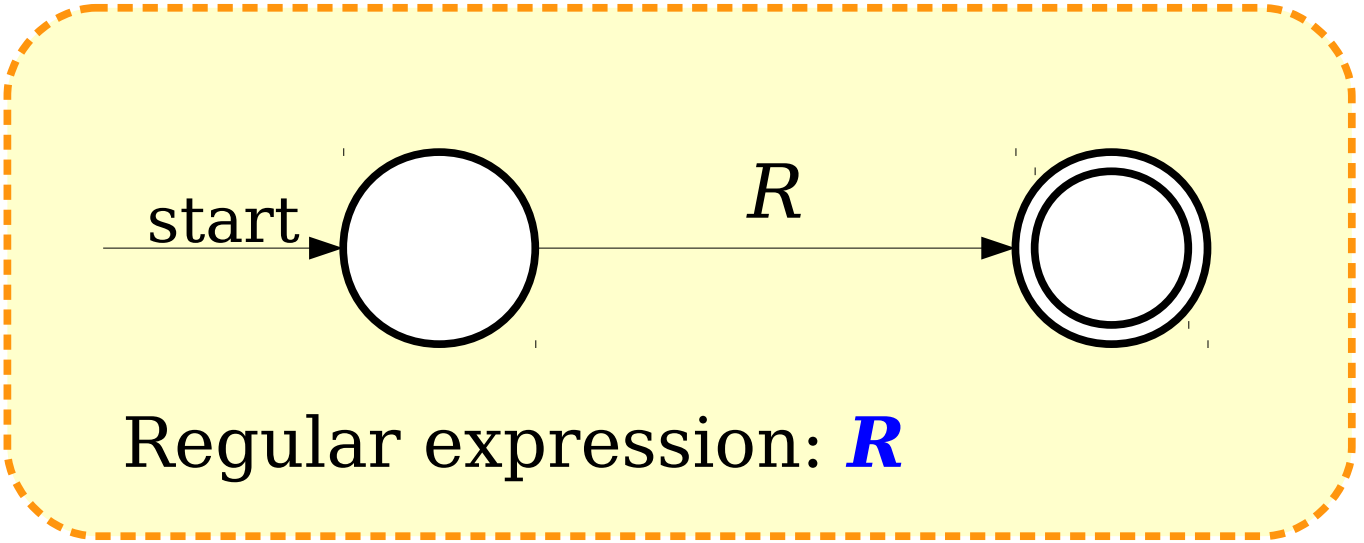


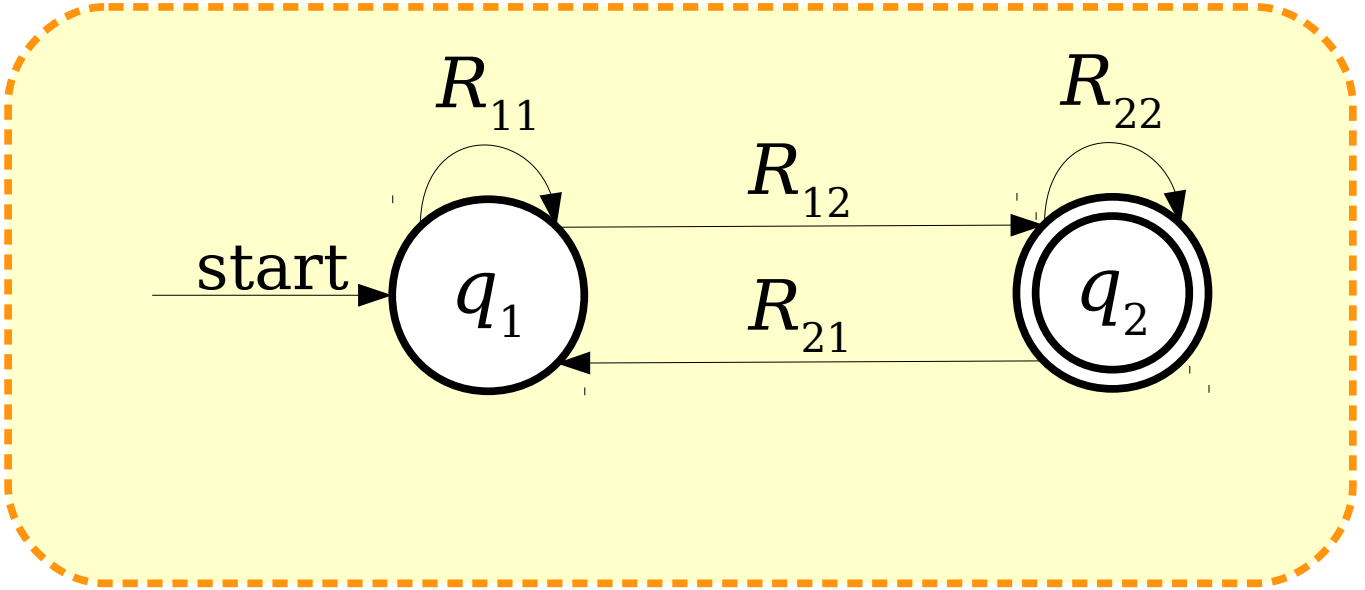


From NFAs to Regular Expressions

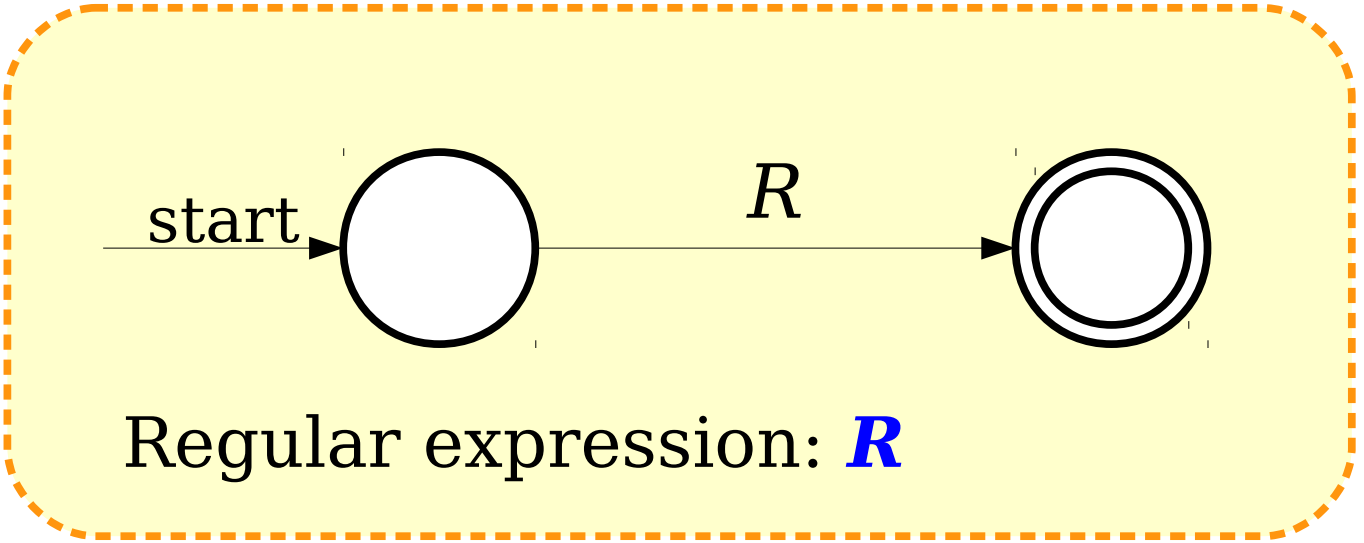


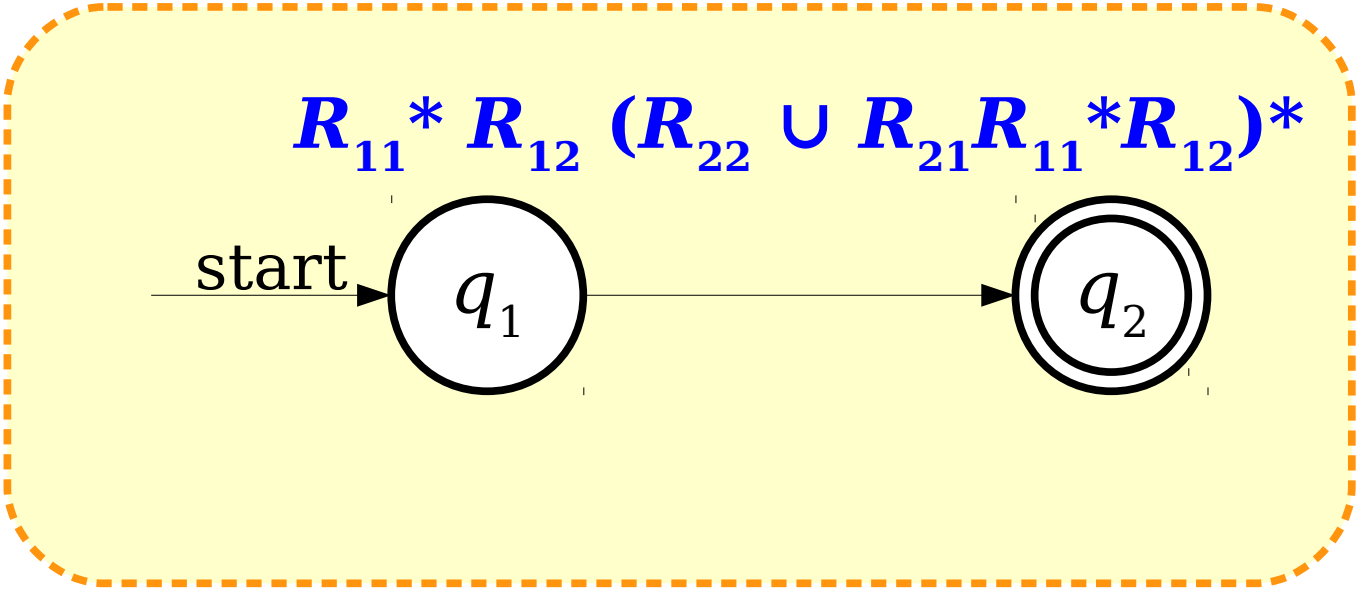
From NFAs to Regular Expressions



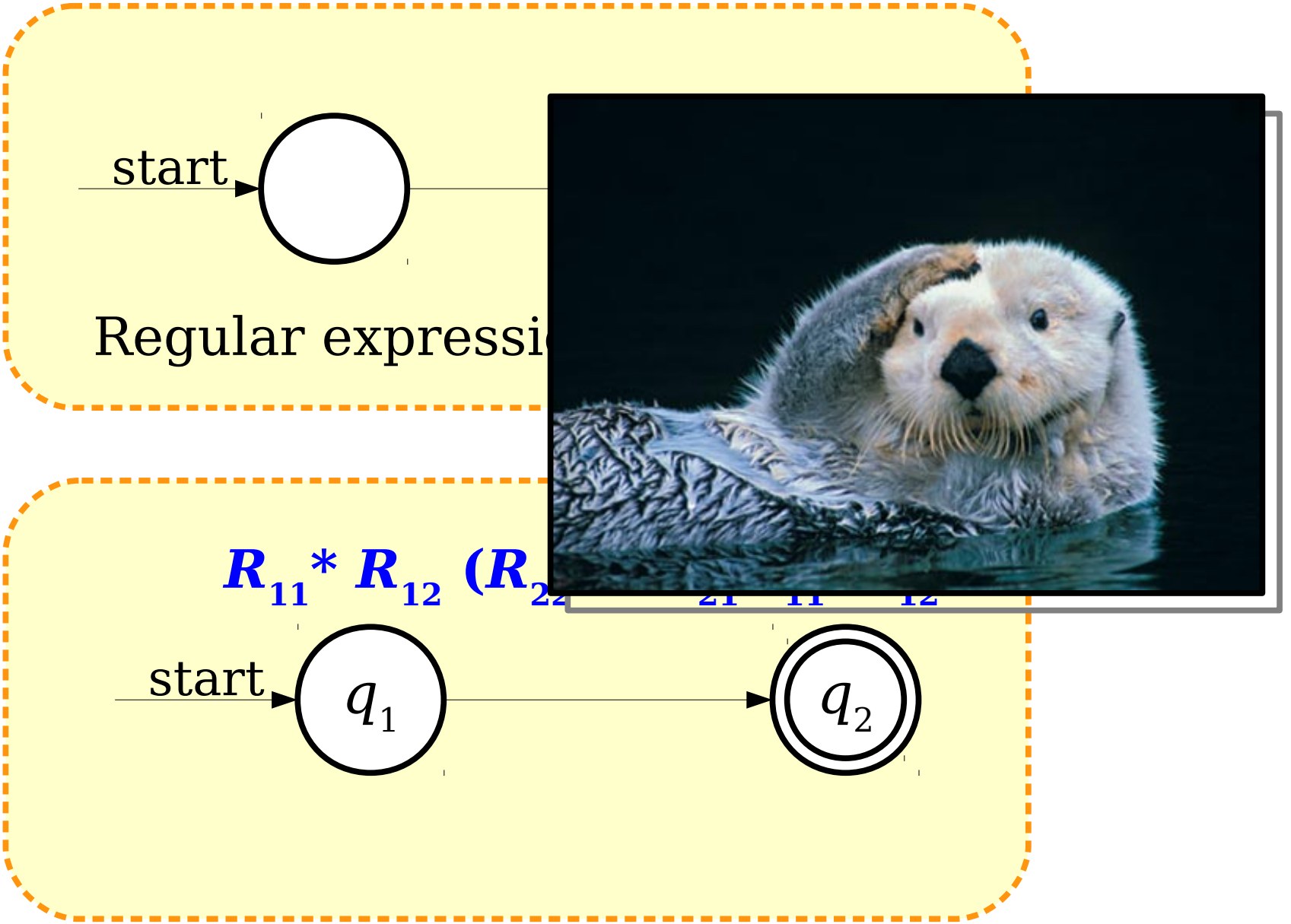


From NFAs to Regular Expressions

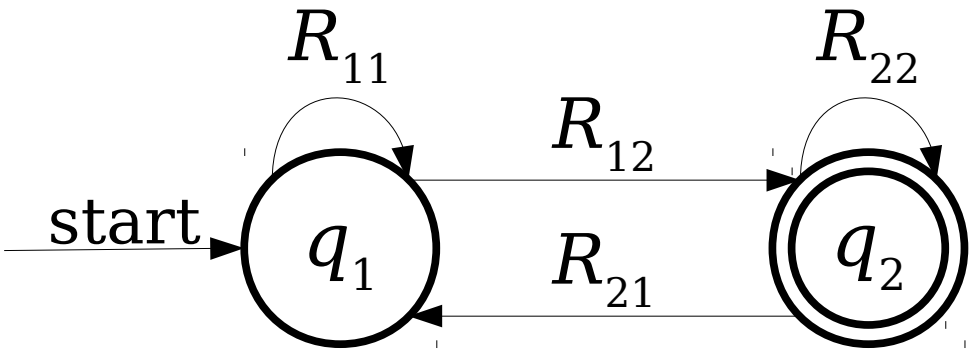




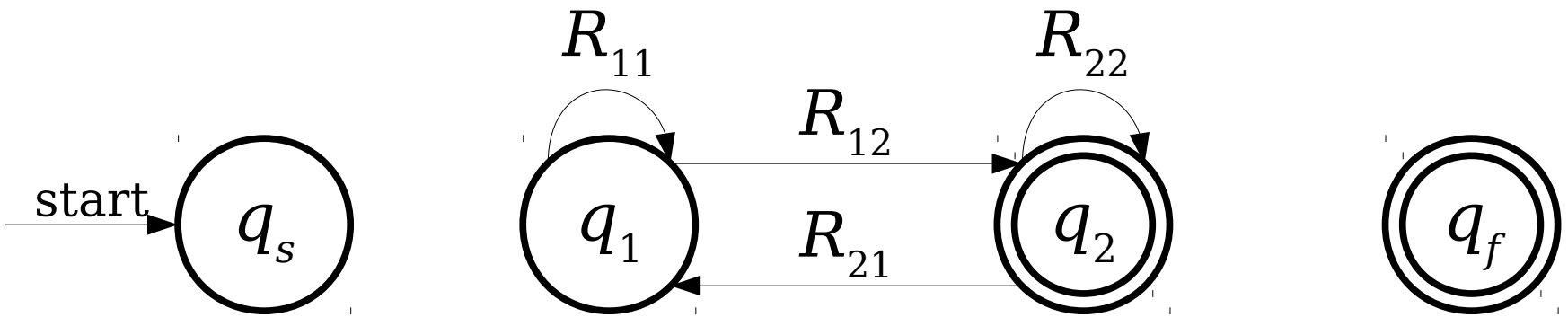
From NFAs to Regular Expressions



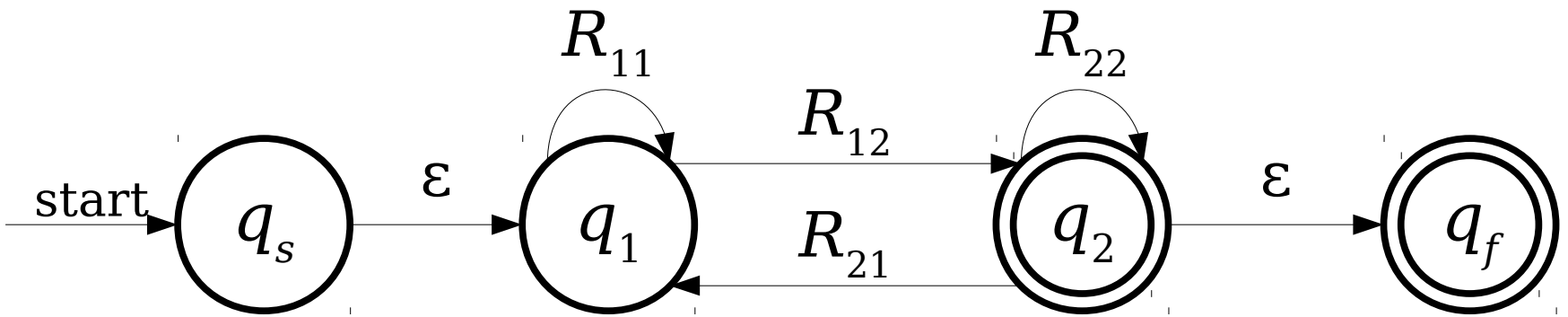
From NFAs to Regular Expressions



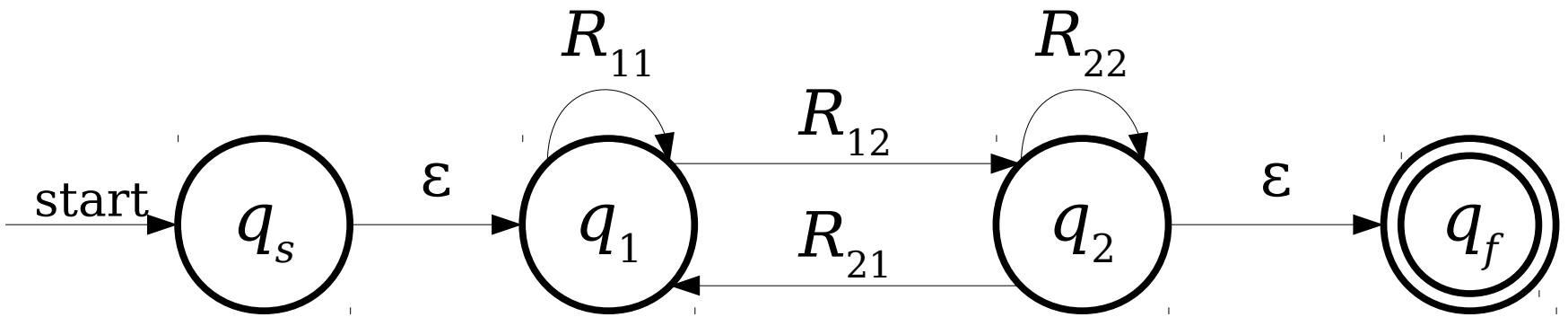
From NFAs to Regular Expressions



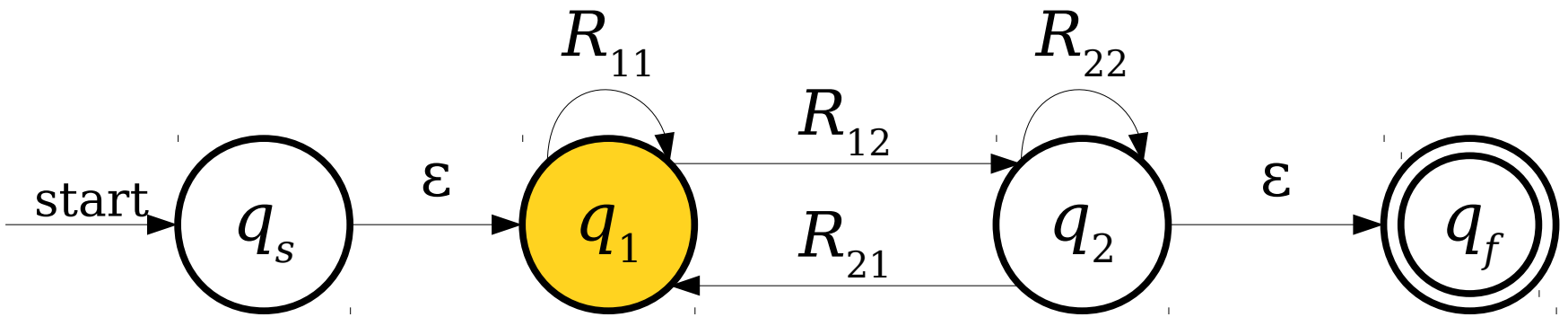
From NFAs to Regular Expressions



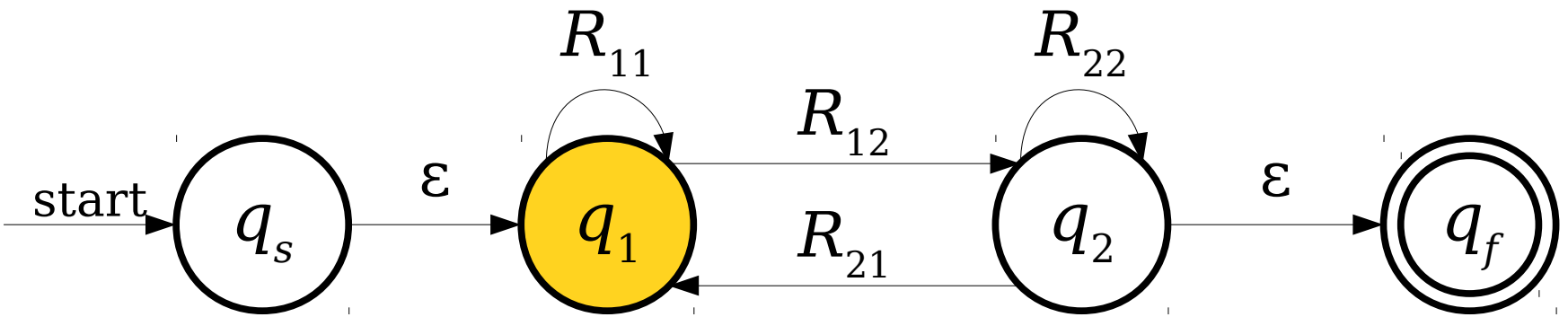
From NFAs to Regular Expressions



From NFAs to Regular Expressions

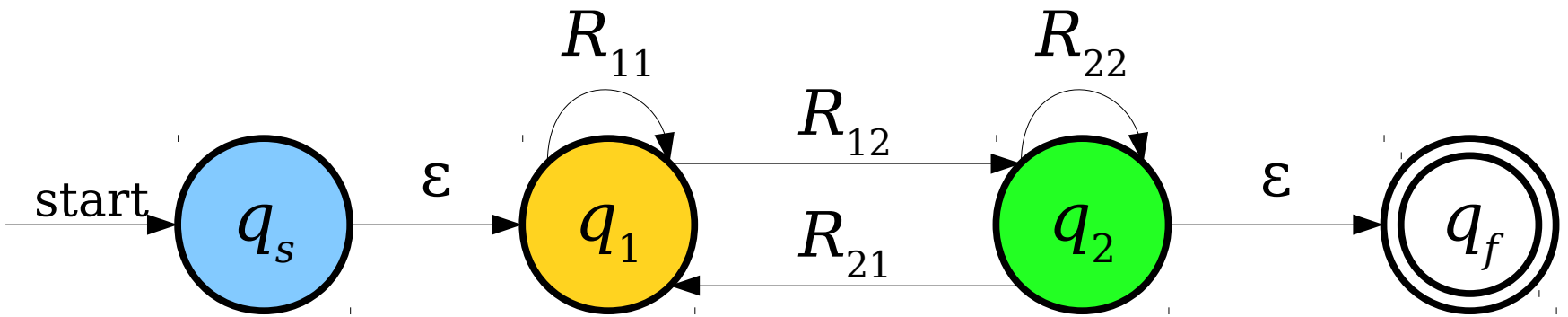


From NFAs to Regular Expressions

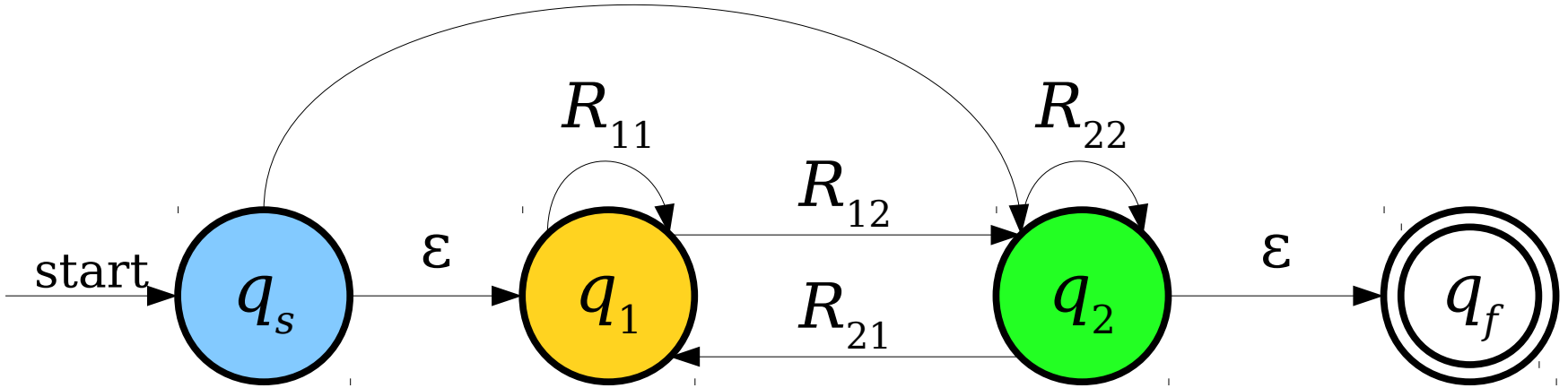


Could we eliminate this state from the NFA?

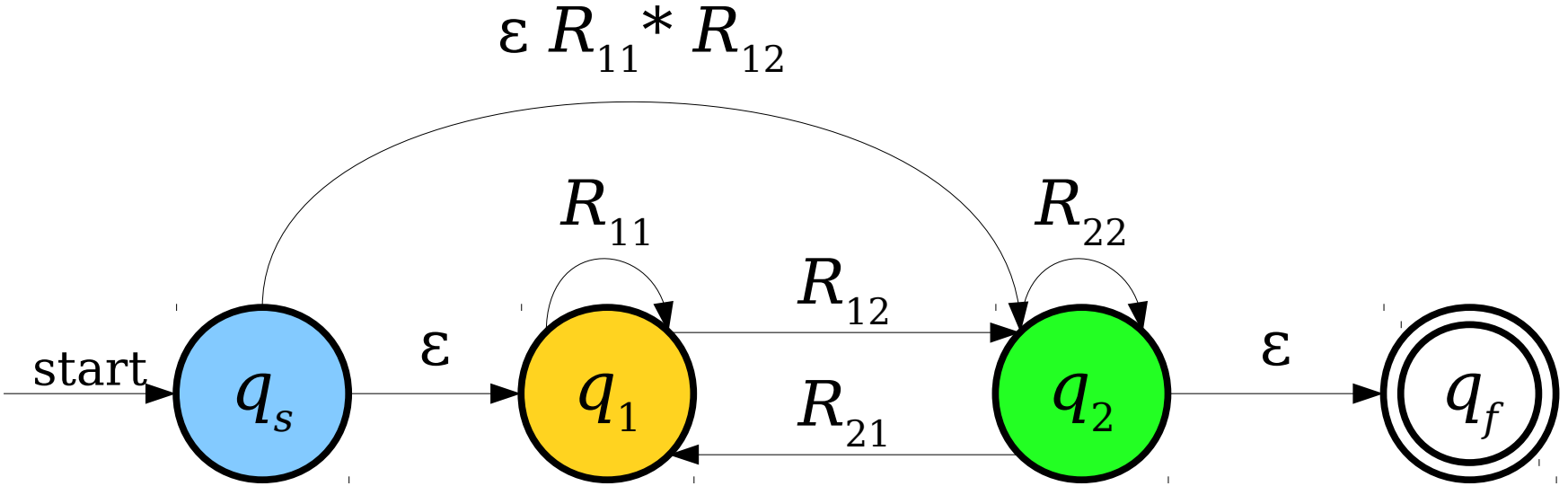
From NFAs to Regular Expressions



From NFAs to Regular Expressions

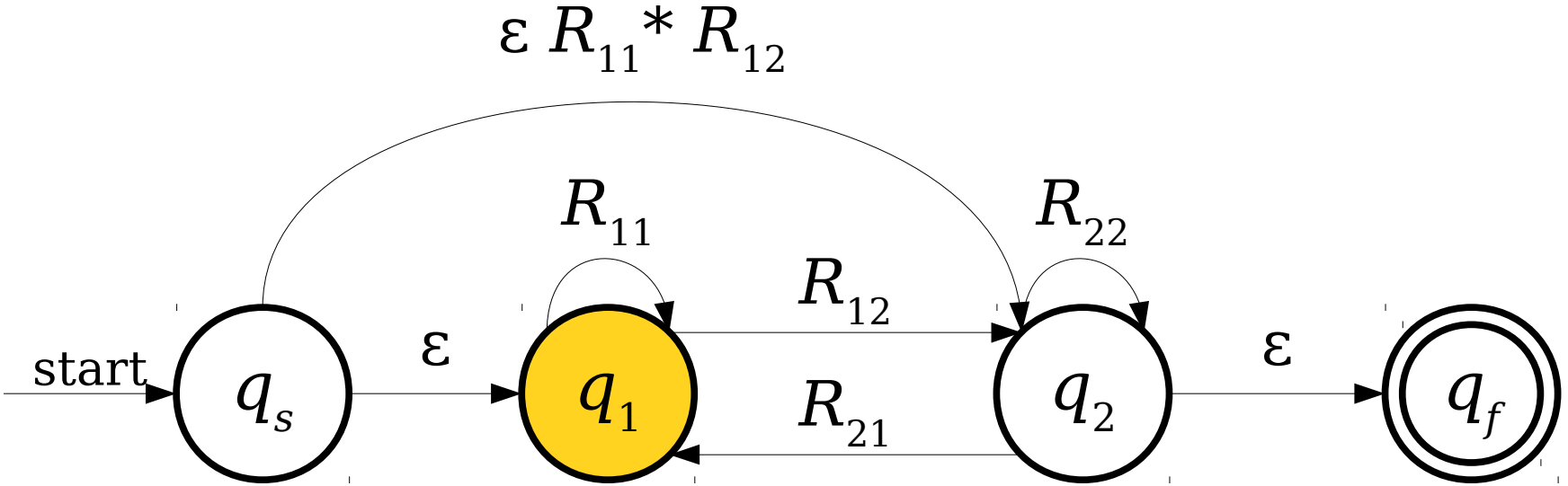


From NFAs to Regular Expressions

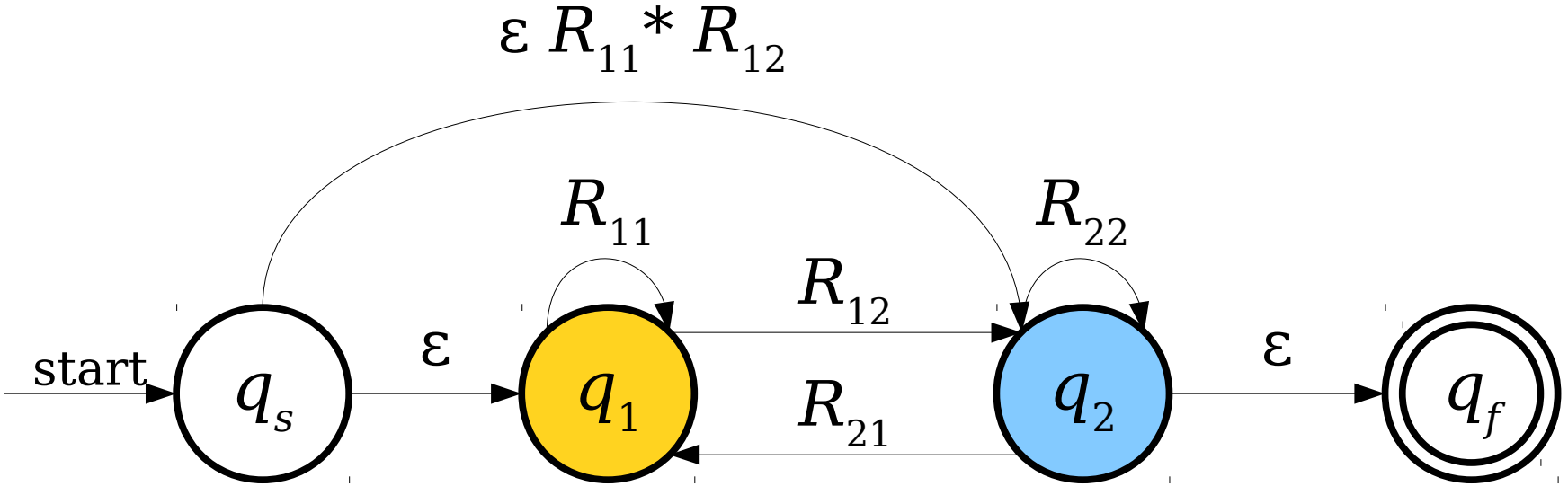


Note: We're using concatenation and Kleene closure in order to skip this state.

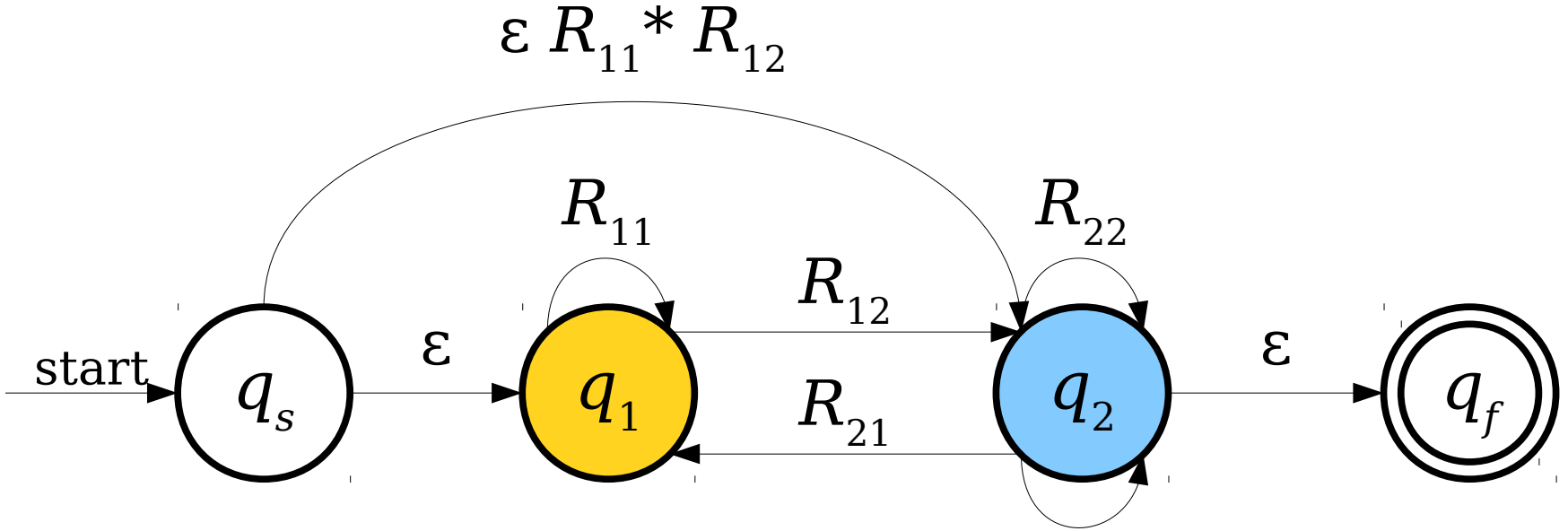
From NFAs to Regular Expressions



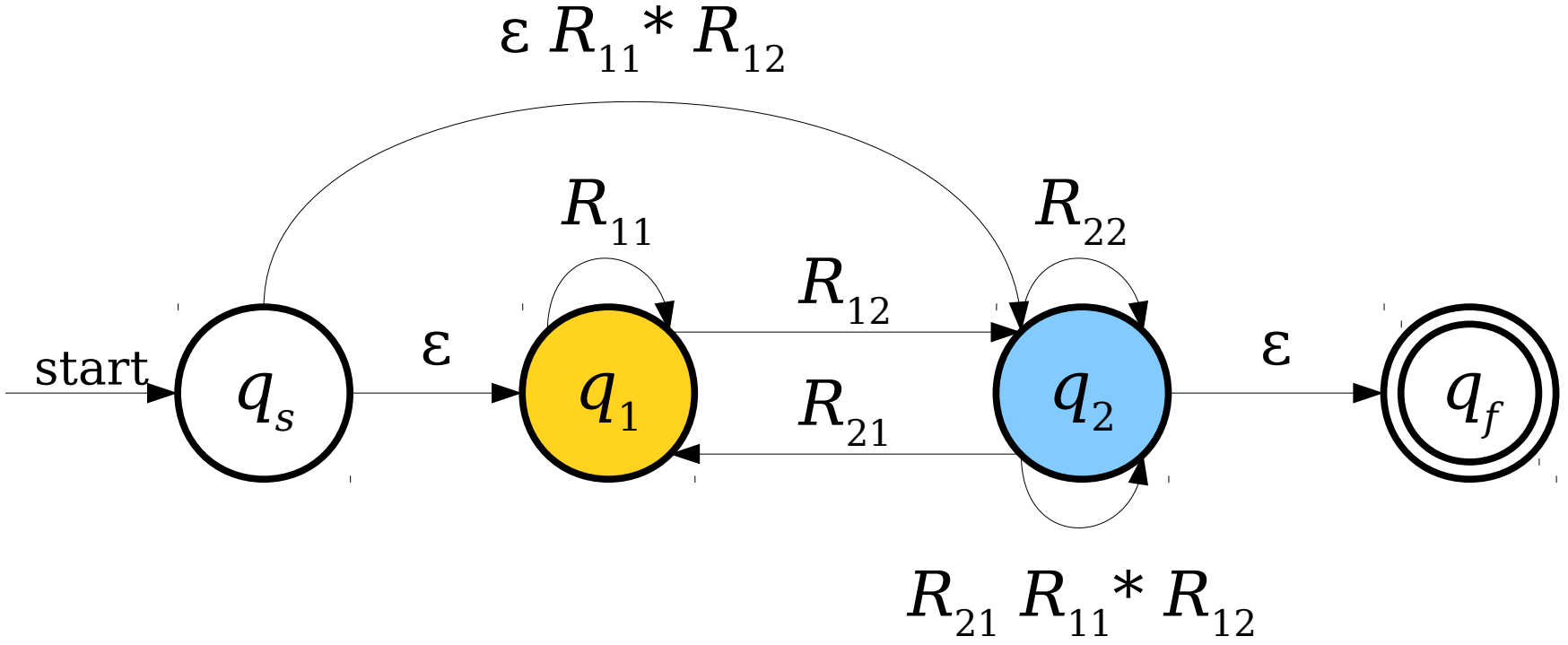
From NFAs to Regular Expressions



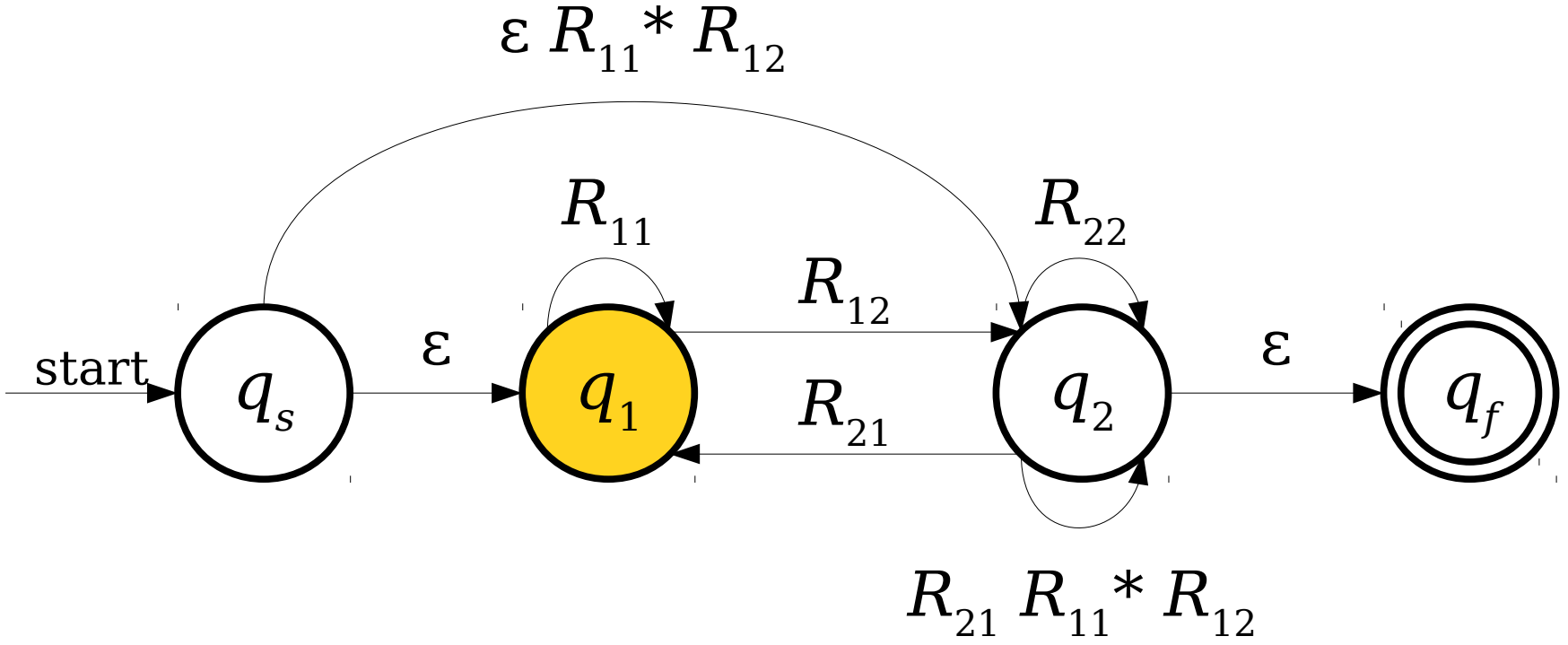
From NFAs to Regular Expressions



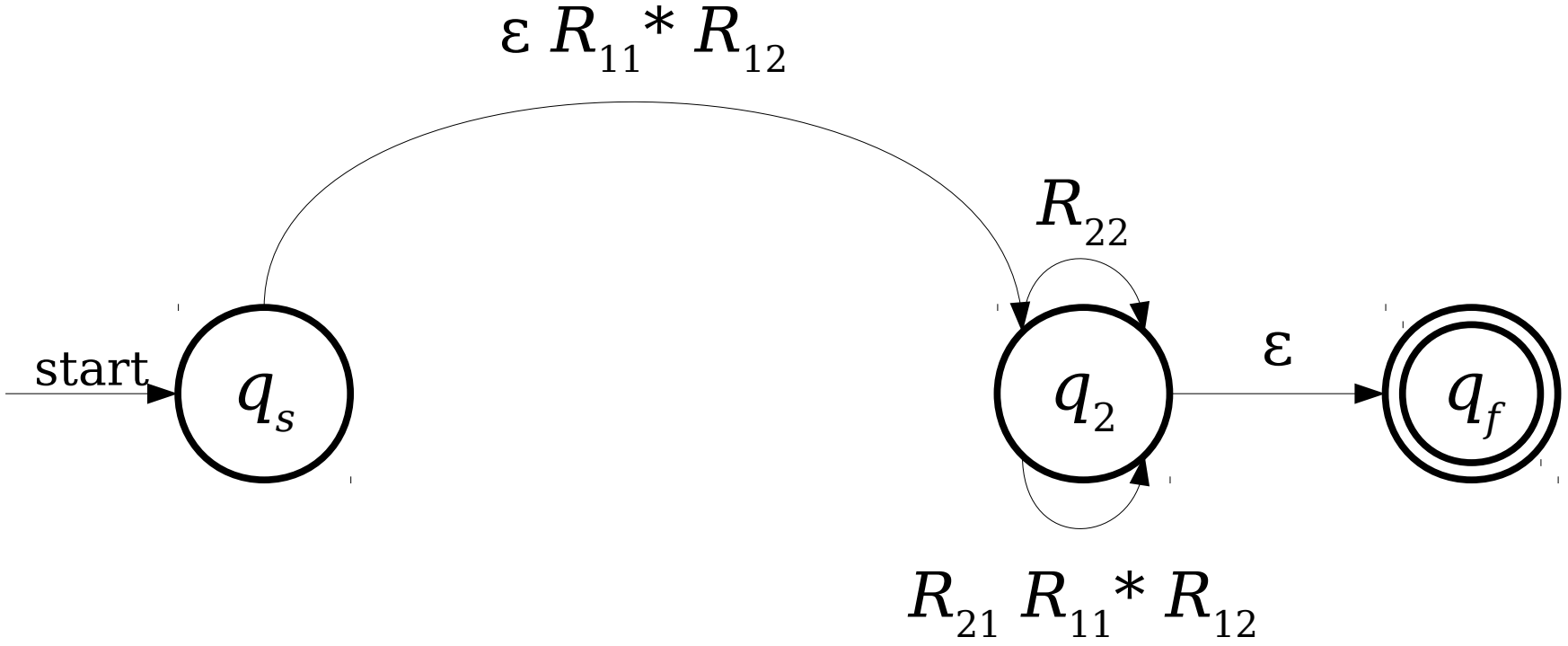
From NFAs to Regular Expressions



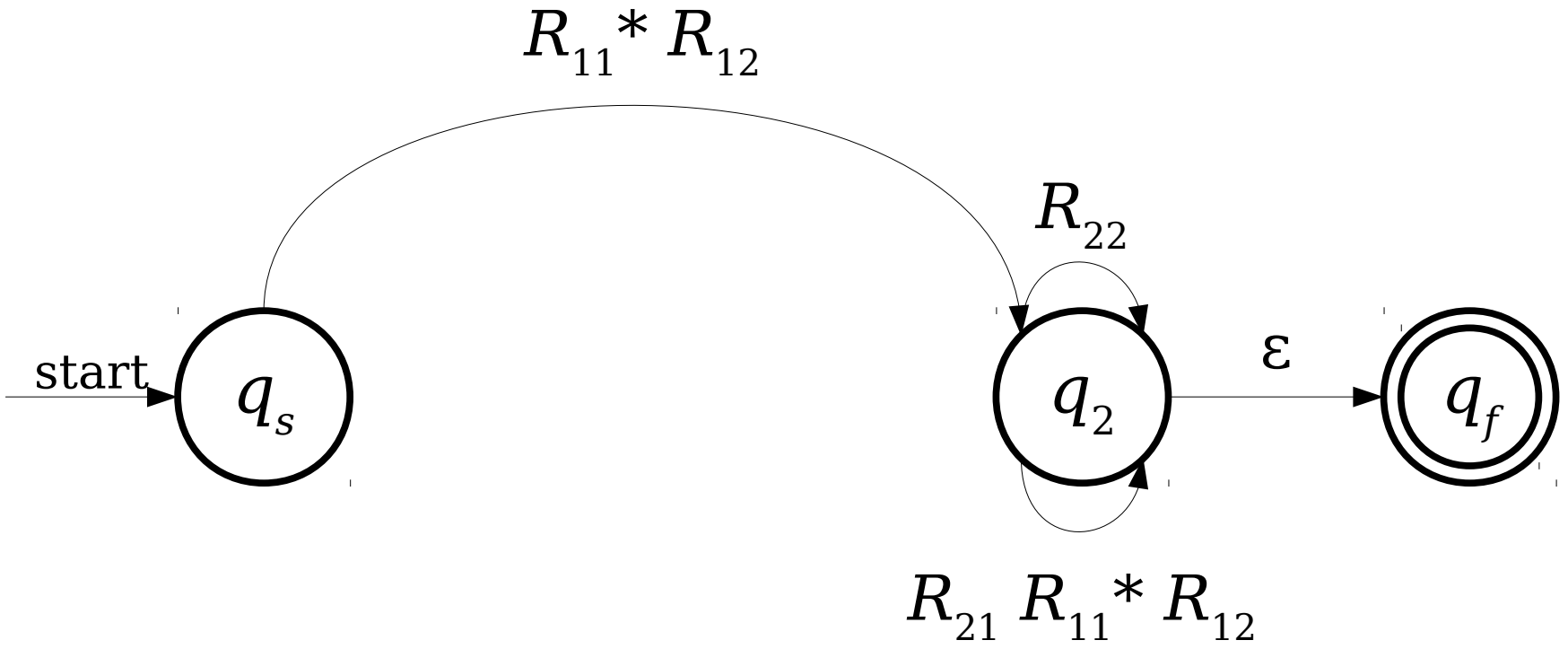
From NFAs to Regular Expressions



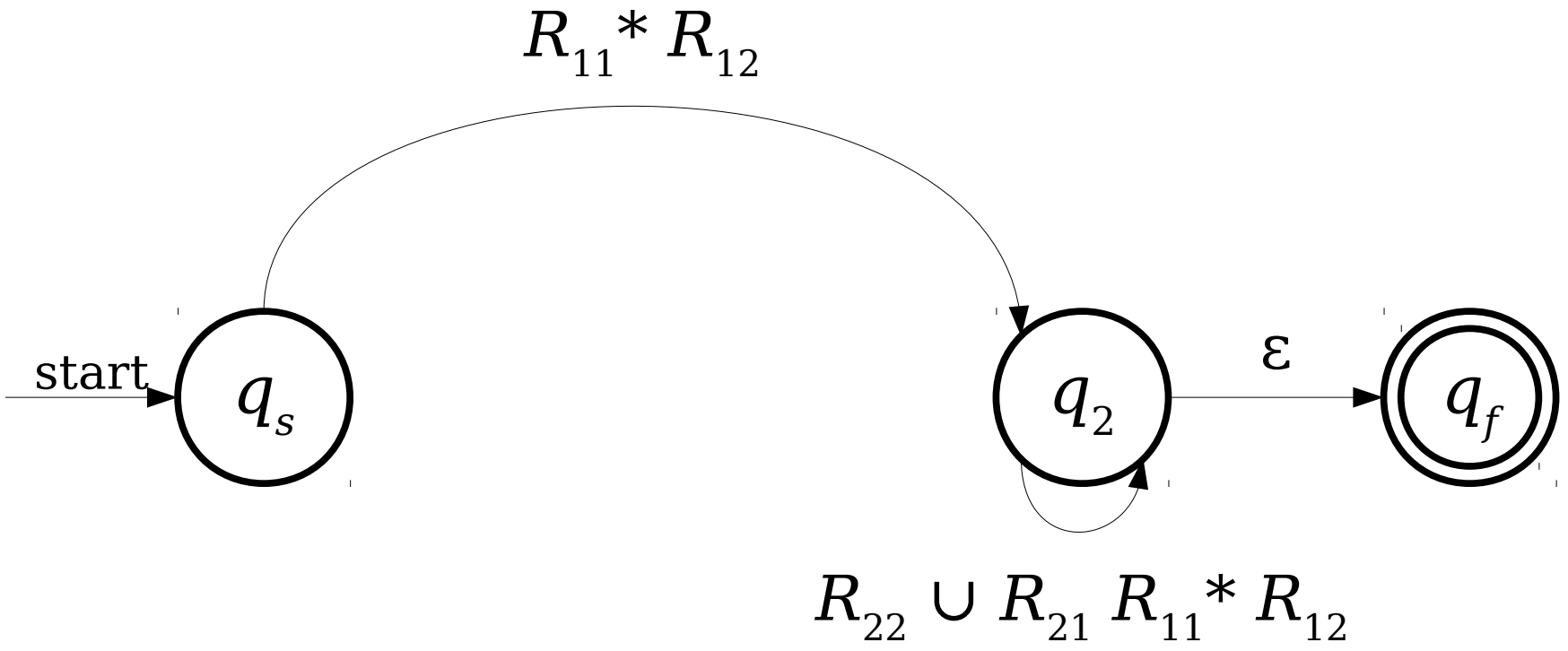
From NFAs to Regular Expressions



From NFAs to Regular Expressions

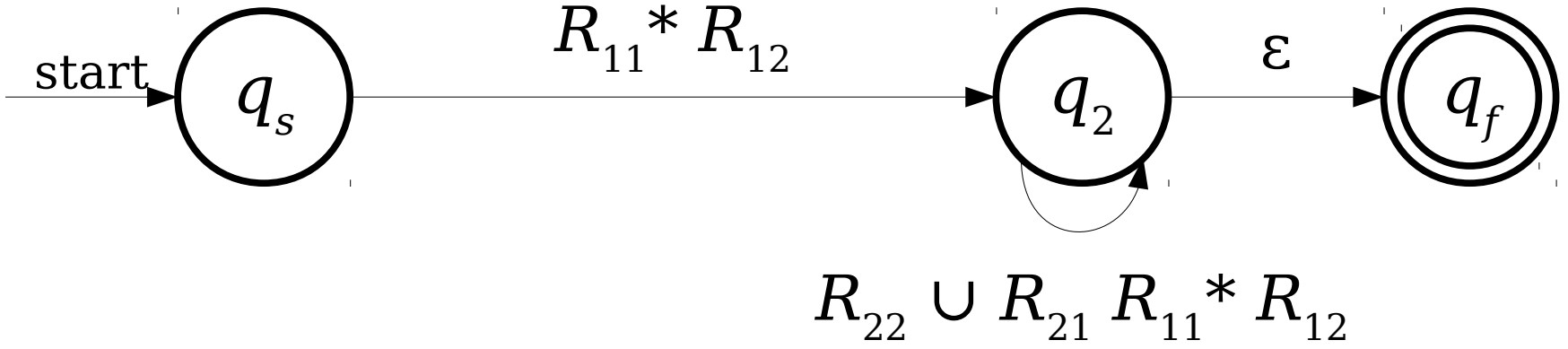


From NFAs to Regular Expressions

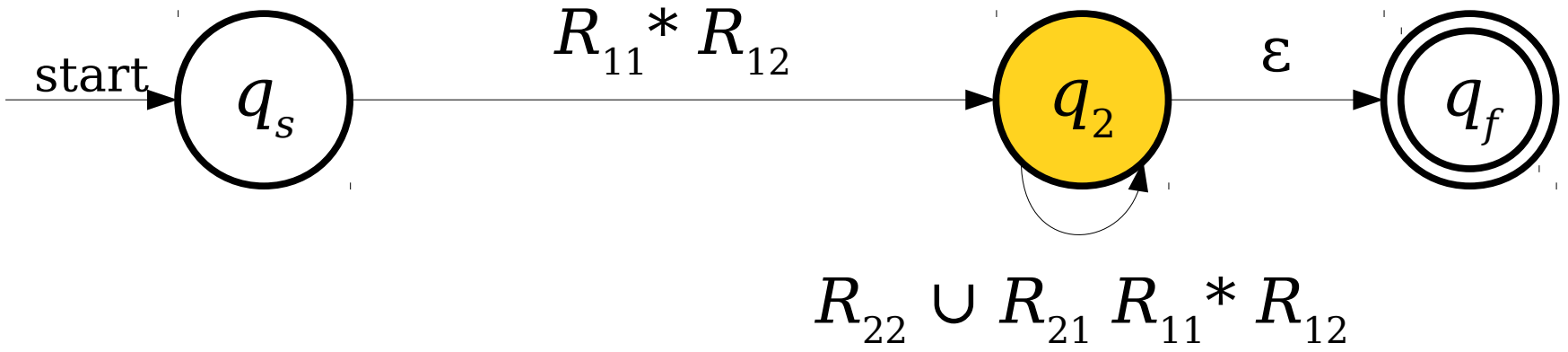


Note: We're using union to combine these transitions together.

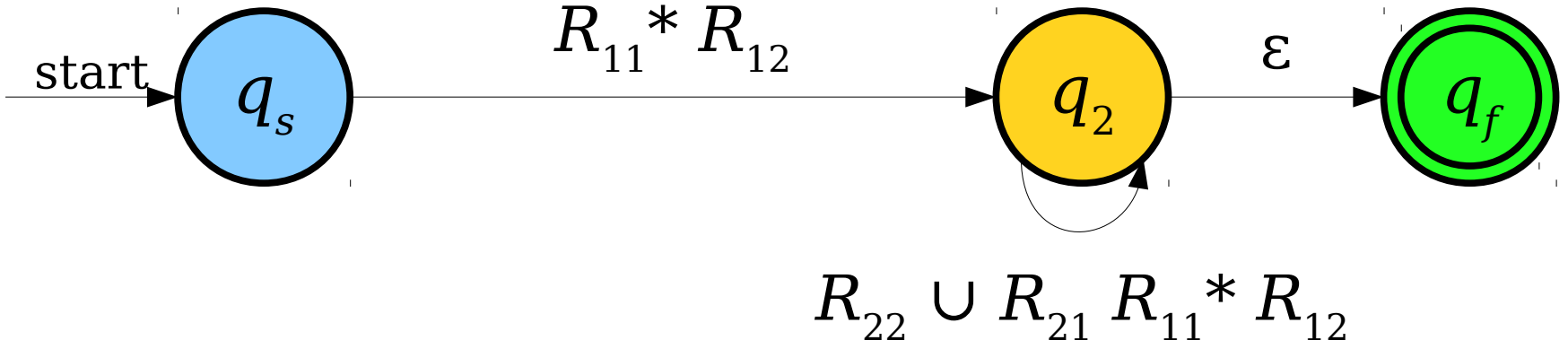
From NFAs to Regular Expressions



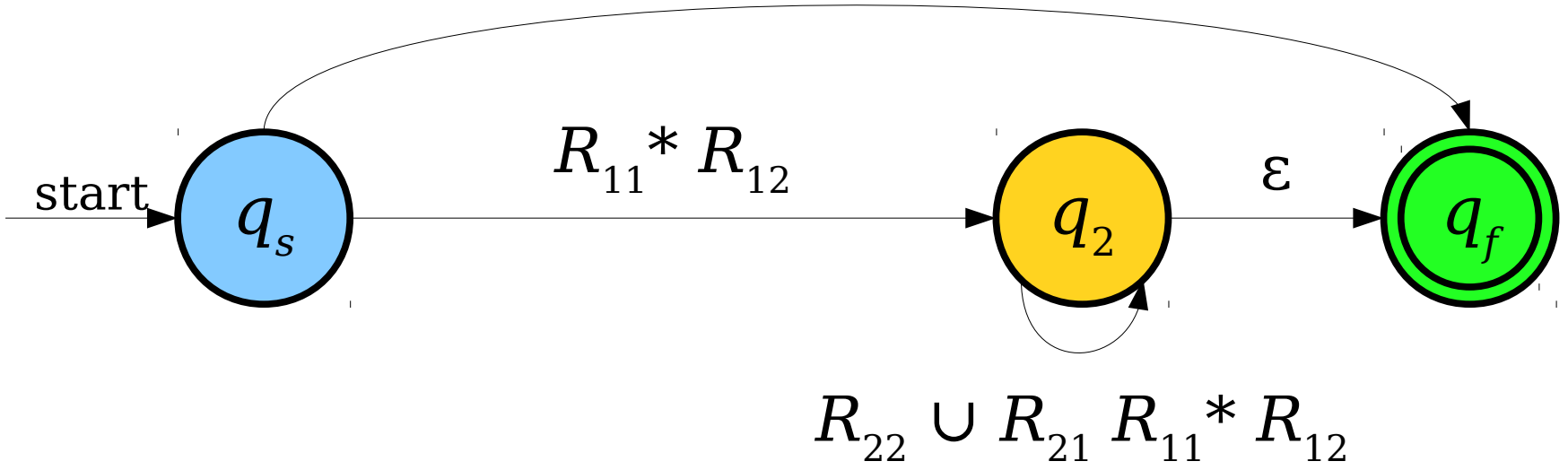
From NFAs to Regular Expressions



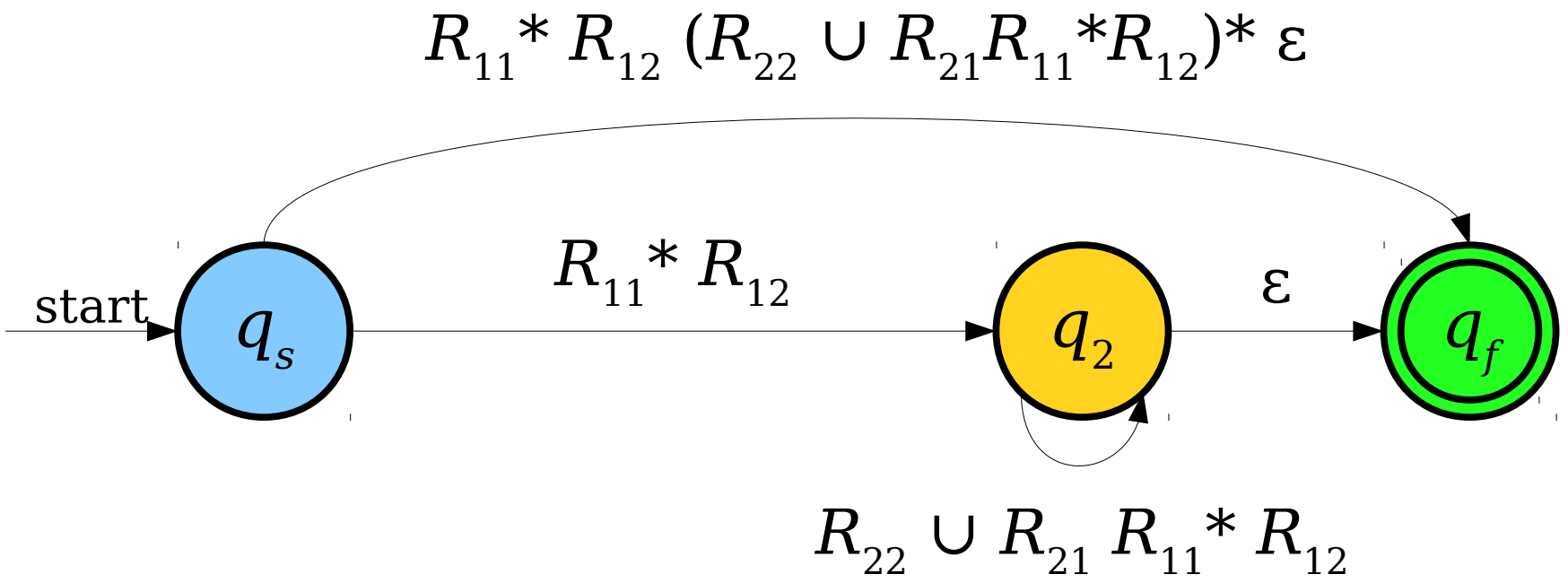
From NFAs to Regular Expressions



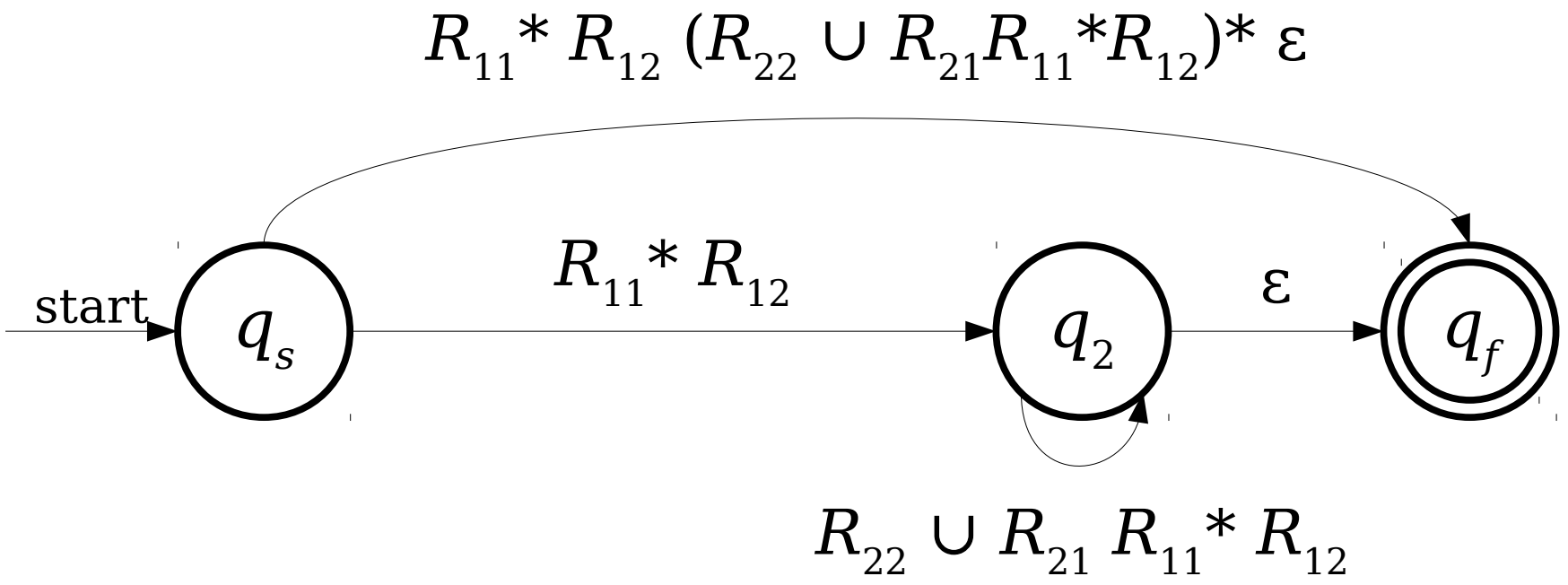
From NFAs to Regular Expressions



From NFAs to Regular Expressions

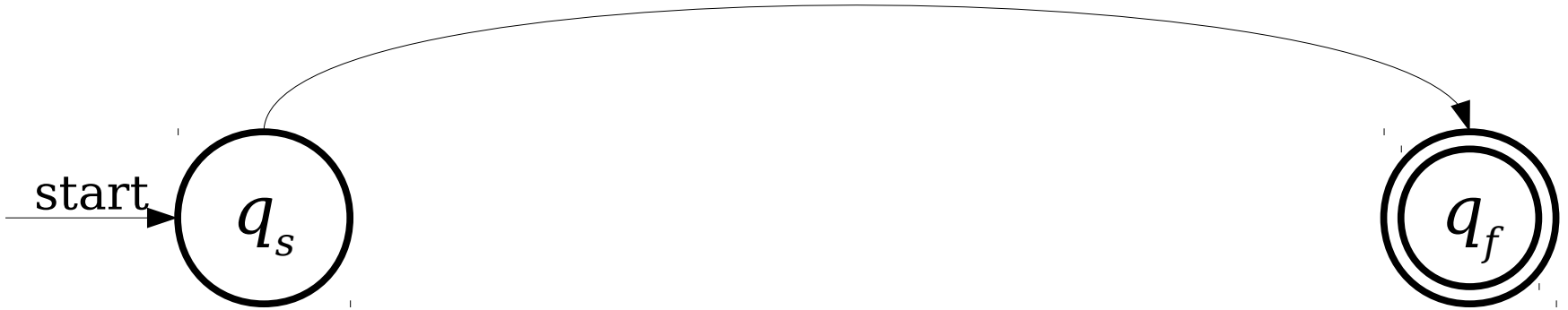


From NFAs to Regular Expressions

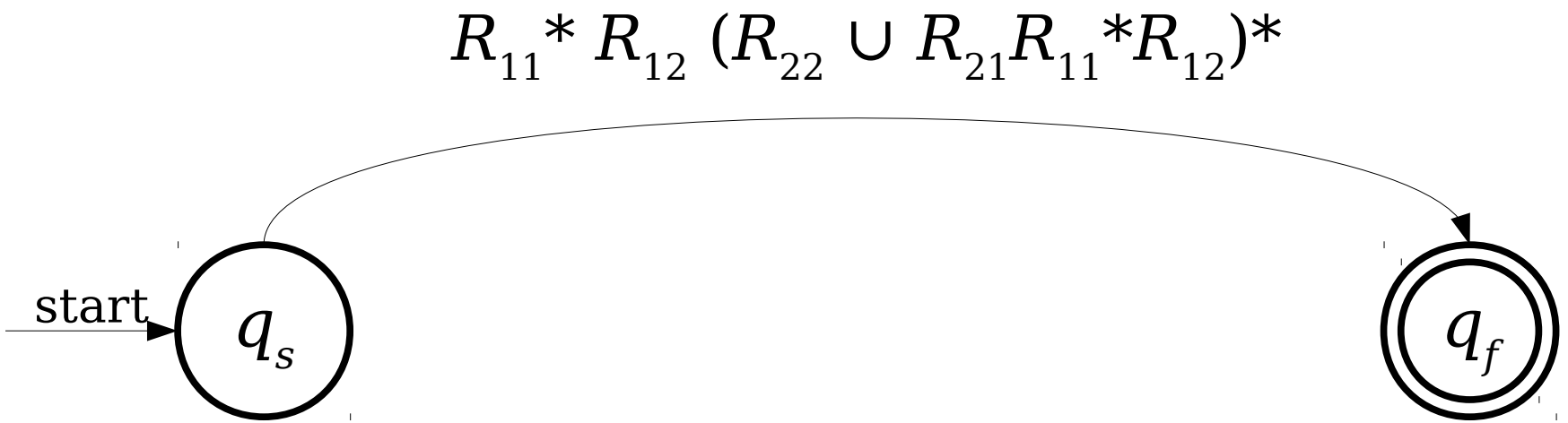


From NFAs to Regular Expressions

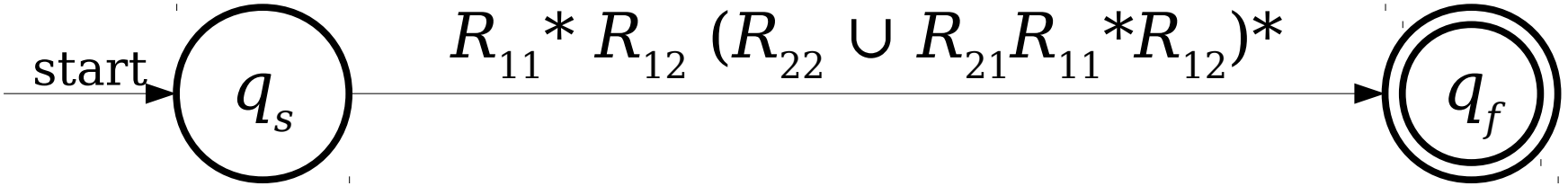
$$R_{11}{}^* R_{12} (R_{22} \cup R_{21} R_{11}{}^* R_{12})^* \varepsilon$$



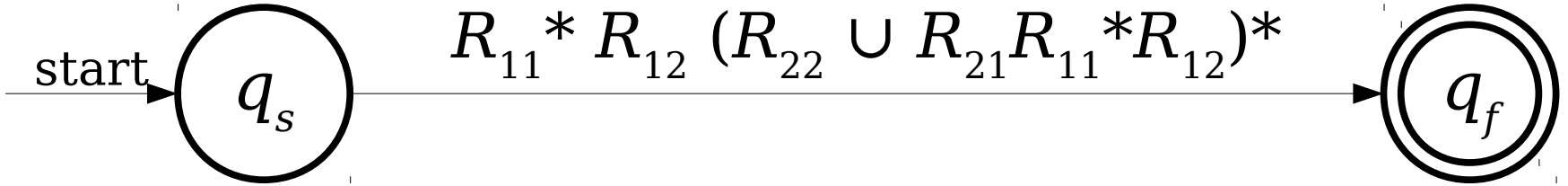
From NFAs to Regular Expressions

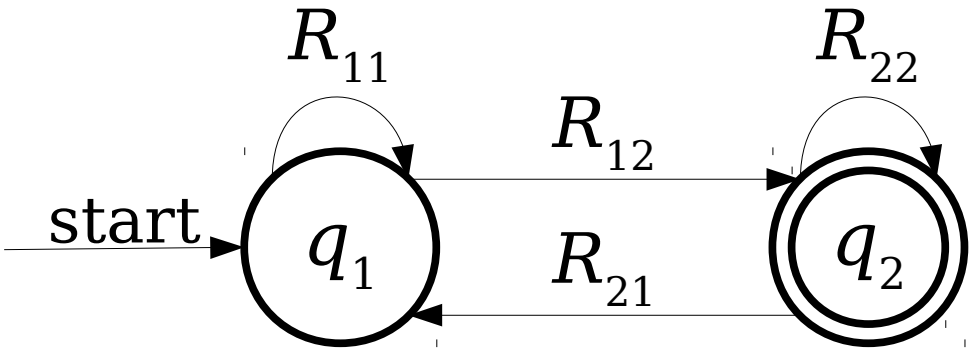


From NFAs to Regular Expressions



From NFAs to Regular Expressions





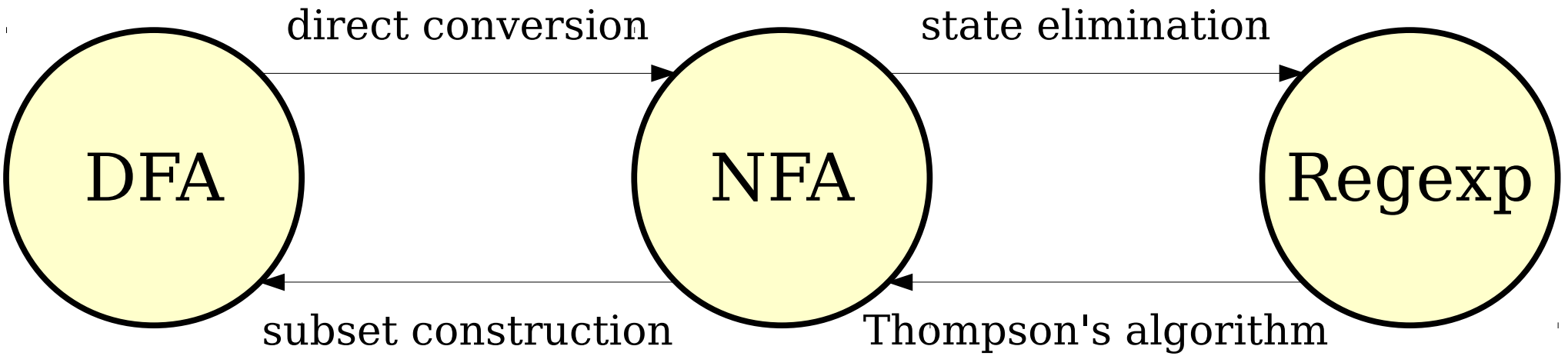
The Construction at a Glance

- Start with an NFA for the language L.
- Add a new start state q_s and accept state q_f to the NFA.
  - Add ε-transitions from each original accepting state to q_f, then mark them as not accepting.
- Repeatedly remove states other than q_s and q_f from the NFA by "shortcutting" them until only two states remain: q_s and q_f.
- The transition from q_s to q_f is then a regular expression for the NFA.

Eliminating a State

- To eliminate a state q from the automaton, do the following for each pair of states q_0 and q_1, where there's a transition from q_0 into q and a transition from q into q_1:
  - Let R_{in} be the regex on the transition from q_0 to q.
  - Let R_{out} be the regex on the transition from q to q_1.
  - If there is a regular expression R_{stay} on a transition from q to itself, add a new transition from q_0 to q_1 labeled $R_{in}(R_{stay})^*R_{out}$.
  - If there isn't, add a new transition from q_0 to q_1 labeled $R_{in}R_{out}$.
- If a pair of states has multiple transitions between them labeled R_1, R_2, …, R_k, replace them with a single transition labeled $R_1 \cup R_2 \cup \ldots \cup R_k$.

Our Transformations



Theorem: The following are all equivalent:

- L is a regular language.
- There is a DFA D such that $\mathcal{L}(D) = L$.
- There is an NFA N such that $\mathcal{L}(N) = L$.
- There is a regular expression R such that $\mathcal{L}(R) = L$.

Why This Matters

- The equivalence of regular expressions and finite automata has practical relevance.
  - Tools like `grep` and `flex` that use regular expressions capture all the power available via DFAs and NFAs.
- This also is hugely theoretically significant: the regular languages can be assembled "from scratch" using a small number of operations!

Next Time

- **Applications of Regular Languages**
  - Answering “so what?”
- **Intuiting Regular Languages**
  - What makes a language regular?
- **The Myhill-Nerode Theorem**
  - The limits of regular languages.